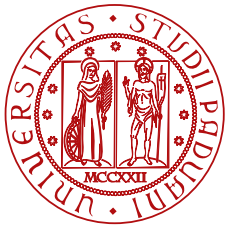


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master's Degree in Computer Engineering

AutoML for Advanced Monitoring in Digital Manufacturing and Industry 4.0

Master Candidate

Alessandro Peratoner

Student ID 2013381

Supervisor

Prof. Gian Antonio Susto

University of Padova

Co-supervisor

Ph.D. Chiara Masiero

Statwolf Data Science Srl

Academic Year 2022/2023

Graduation Date 2023 February 27°

To my family,
and friends

Abstract

The emergence of Industry 4.0 and the associated rise of sensing technologies in industrial equipment has made the adoption of Machine Learning (ML) solutions crucial in enhancing and making more efficient enterprise production processes. However, the high demand for ML models often clashes with the small number of professionals capable of handling such projects. For this reason, Automatic Machine Learning (AutoML) tools are considered high-value solutions, thanks to their capability to provide a suitable model for the provided data without the need for the intervention by a ML expert. Indeed, AutoML libraries are designed to be used in an easy way also for people with limited or no experience with ML.

In this work, three important tasks involved in manufacturing, that are Anomaly Detection, Visual Anomaly Detection, and Remaining Useful Life Estimation, are considered. After analysing the most critical aspects of each task and the state of the art, possible solutions are proposed for the development of specialised AutoML modules. Additionally, given the increasing emphasis on the interpretability of ML models, part of the analysis performed aims at identifying explainability tools, which are particularly important for an AutoML library. In fact, they provide useful motivations for the model predictions, increasing also the user confidence in AutoML tools.

Contents

List of Figures

List of Tables

List of Acronyms

ABOD Angle-Based Outlier Detector

AcME Accelerated Model-agnostic Explanations

AD Anomaly Detection

AE Auto-Encoder

ANN Artificial Neural Network

AUPR Area Under the Precision-Recall curve

AUROC Area Under the Receiver Operating Characteristic curve

AutoML Automated Machine Learning

CASH Combined Algorithm Selection and Hyper-parameter

CL Custom Loss

CNN Convolutional Neural Network

CPPS Cyber-Physical Production Systems

CPS Cyber-Physical Systems

CV Computer Vision

DL Deep Learning

DNN Deep Neural Network

GAN Generative Adversarial Network

GB Gradient Boosting

HBOS Histogram-Based Outlier Score

HITL Human In The Loop

IF Isolation Forest

IIoT Industrial Internet of Things

IoT Internet of Things

LIME Local Interpretable Model-agnostic Explanations

LODA Lightweight On-line Detector of Anomalies

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

ML Machine Learning

MLP Multilayer Perceptron

MSE Mean Squared Error

NF Normalizing Flow

NN Neural Network

OD Outlier Detection

PCA Principal Components Analysis

PdM Predictive Maintenance

PIADE Packaging Industry Anomaly Detection

PM Preventive Maintenance

R2F Run-To-Failure

ReLU Rectified Linear Unit

RF Random Forest

RNN Recurrent Neural Network

RUL Remaining Useful Life

SME Small and Medium-sized Enterprise

WMSE Weighted Mean Squared Error

1 Introduction

1.1 Industry 4.0

1.1.1 The Fourth Industrial Revolution

In recent years, the debate regarding Industry 4.0 has become increasingly important. The term Industry 4.0 dates back to November 2011 and was introduced by the German government as "Industrie 4.0" at the Hanover Trade Fair. In the following years, this concept attracted the attention of many other governments, especially in Europe, that began to develop local programs to implement this new approach [15] [22].

Often this concept is associated with a real industrial revolution, that is the fourth one. Looking back at the history of the previous three industrial revolutions, which spanned almost 200 years, we can see that they always coincided with revolutions in technology and approach to manufacturing. The First Industrial Revolution occurred during the second half of the 18th century and it was characterized by the introduction of the steam engine and the flying shuttle, as well as the exploitation of water power and the passage from hand production methods to mechanization. The assembly lines, pioneered by Henry Ford, are the main protagonists of the Second Industrial Revolution, which came up during the second half of the 19th century. The replacement of steam with chemical and electrical energy, together with the first use of petroleum, contributed to the development of mass production. During the second half of the previous century, the invention of the Integrated Circuit (microchip) was the technological

driver of the so-called Third Industrial Revolution, characterized by the usage of electronics and Information Technology to further increase automation in production [42] [22] [15].

Similarly to the previous revolutions, also Industry 4.0 includes some technological innovations. Indeed, it can be seen as an umbrella term for several technologies, some of which are sometimes considered synonyms of Industry 4.0: Internet of Things (IoT), Cyber-Physical Systems (CPS), Cloud Computing, Big Data, Augmented Reality, Autonomous Robotics, Additive Manufacturing, and many others [15] [42]. The increasing popularity of Industry 4.0 and some related concepts in the literature is visible in Figure 1.1, taken from [15].

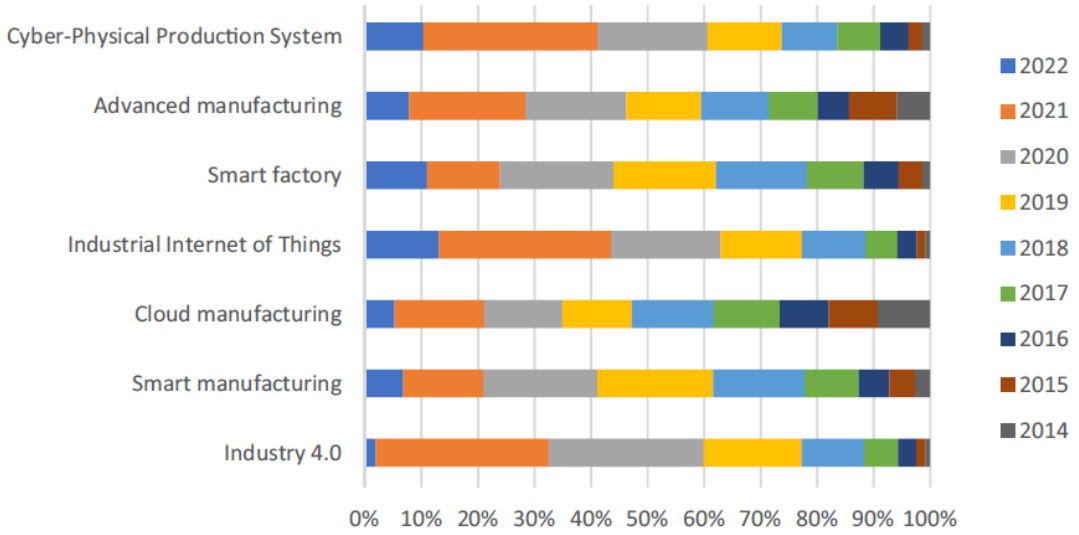


Figure 1.1: Popularity of Industry 4.0 and other related concepts based on the number of publications of each concept in the last seven years (up to May 2022). Extracted from the Scopus database using exact match in the title [15].

Despite its increasing importance in literature, there is no precise and generally accepted definition of Industry 4.0 among researchers [15]. However, the most complete is provided by a recent systematic review [15], which defines Industry 4.0 as a "manufacturing philosophy that includes the application of digital technologies in internal and external manufacturing operations in a way that enables real-time integration (vertical, horizontal, end-to-end) among all participants of the value chain to enhance operations and improve competitiveness". Beyond this definition, there exist some design principles that characterise Industry 4.0 and on which many researchers agree [15] [22] [42] [35]:

- **Interoperability**: the capability of systems to transact with other systems [22]. In Industry 4.0 all factory components, i.e., machines, robots, people,

and objects are connected through IoT and can communicate and share different kinds of data.

- **Virtualisation**: in Industry 4.0 the physical and virtual worlds are interconnected and a virtual copy of everything is created (digital twin)[15]. This is achieved by means of the data collected through the huge amount of sensors embedded in the factory's elements. Virtualisation is particularly useful for maintenance.
- **Real-time capacity**: it is the ability to collect data in real-time from different sources, analyse them, and take actions on the basis of the extracted information, which reflect the real-world conditions.
- **Service orientation**: this principle is also known with the expression Product-as-a-Service (PaaS), and is related to the ability of all connected elements of the system to react to changes in the market and customer demands [15]. The paradigm shift involves treating the product increasingly as a service in order to add value to the customer.
- **Modularity**: Industry 4.0 requires a major paradigm shift in industrial manufacturing, with a transition from linear and rigid manufacturing toward a flexible and modular organisation, particularly able to quickly adapt to changes in customer needs and market requirements.
- **Decentralisation**: the factory's components in Industry 4.0 must be as autonomous as possible and be able to make decisions without intermediaries. At the same time, they must be aligned with the final goal of the whole system.
- **System integration**: there are three types of integration. Vertical integration, also called intra-company, involves the units of a single organisation; horizontal integration, instead, embraces various organisations over the value chain of the product; the end-to-end integration involves the entire product life cycle, creating customised products and services [15].
- **Corporate social responsibility**: this principle involves areas such as environmental and labour regulations. Indeed, companies that want to embrace this new business model have to consider its impact in terms of jobs created or killed, and have to care about the environmental sustainability of the new manufacturing approach [22] [21].

In Figure 1.2, taken from [22], are represented all the key principles of Industry 4.0, together with the enabling technologies.

At the core of Industry 4.0 there is the concept of Smart factory, where all production facilities and machines are connected and share information. Industrial Internet of Things (IIoT), that is, the application of IoT technologies in industrial production, is a technological enabler of this new type of factories [15]. A second key enabler is represented by the Cyber-Physical Production Systems (CPPS). In general, the Cyber-Physical Systems (CPS) are automated

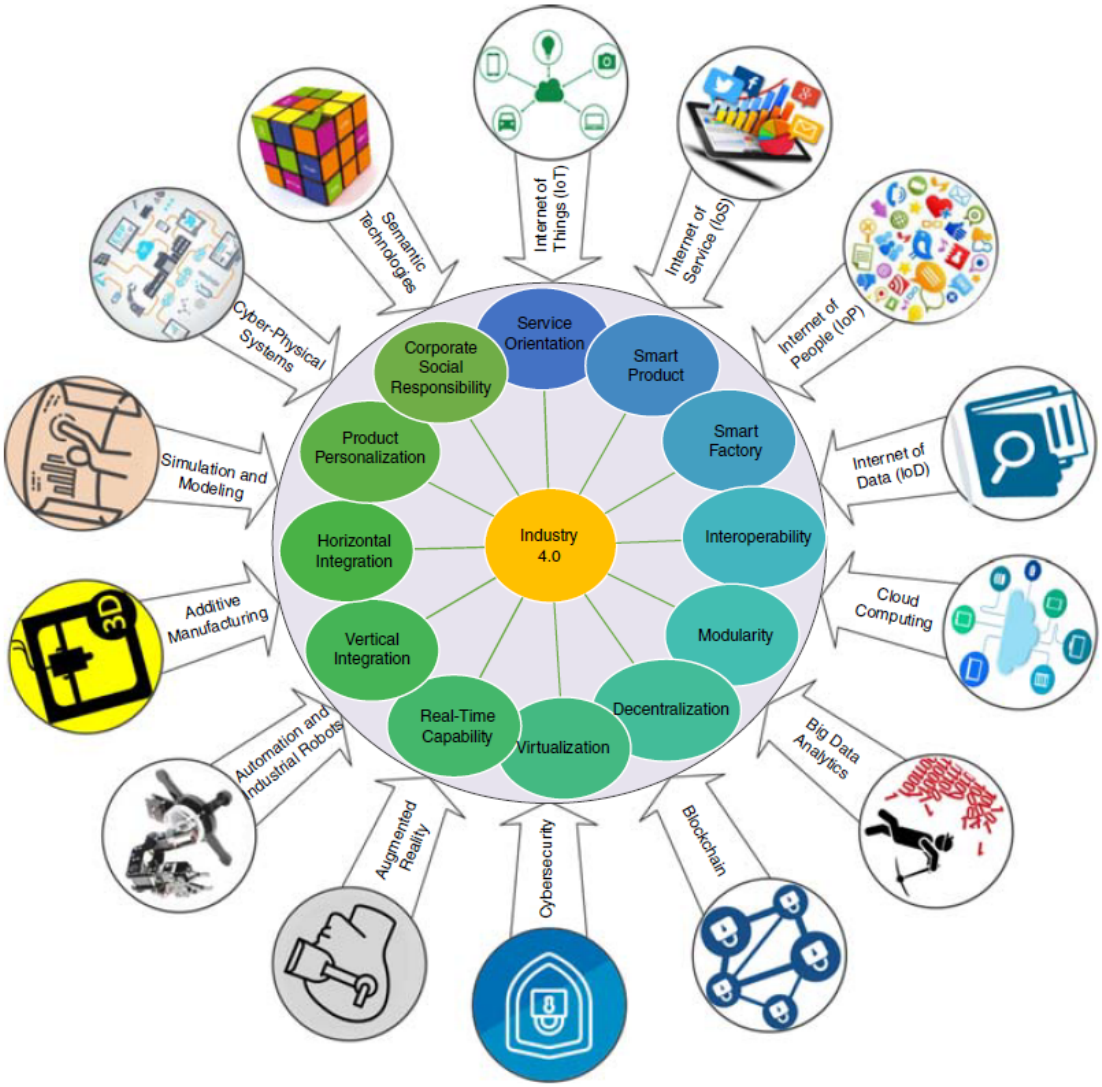


Figure 1.2: Design principles and technology trends of Industry 4.0 [22].

systems characterised by a deep interconnection between the physical and the virtual environments. This is achieved by means of a network of sensors that allow monitoring and extracting data from the physical world and creating a virtual copy of it. Through the use of several smart actuators, this allows to act in the physical environment, according to the information obtained. Embedded systems facilitate the coordination of the physical elements and the digital ones [15]. When this technology is adopted for production, is know as CPPS. They are basically similar to IIoT, but in the latter case CPPS are connected through the Internet [29].

The goal of a Smart Factory is process efficiency, achieved through ma-

chine and equipment automation and self-optimisation. This factory model also allows for the reduction of resource waste, product defects, and machine downtimes [22].

Another important concept is the Smart product, a special kind of product provided with many different kinds of sensors that are embedded in it. These sensors are able to acquire data about the product, the environment, and the customer and communicate them through the Internet. This can be useful both when the product is on the production line, being able to provide information about its status, but also at the consumption stage, where information extracted from sensors data can be considered an added value for the customer [22] [23].

The concepts and principles outlined above are just a few aspects that characterise this Fourth Industrial Revolution. Moreover, the Industry 4.0 topic is not completely defined and subject to debate among experts, also because of the underlying technology which is constantly evolving. From an implementation point of view, then, as noted by [22], to date there is no well-defined roadmap that would suit all companies interested in transitioning to Industry 4.0. There are a lot of challenges that a company has to face, among which there are the huge initial capital investment, the low availability of employees with the needed skills, the risk related to data security and the environmental impact of the transition. However, the benefits of Industry 4.0 could be remarkable, for instance, in terms of revenue gains, cost reduction, improved efficiency and productivity, competitivity, without excluding those in the environmental and social fields [15] [22] [21].

1.1.2 Machine Learning for Industry 4.0

As observed in the previous subsection, the virtual world of digits plays a vital role in Industry 4.0. Indeed, the factory's components continuously collect data about their status and the environment thanks to the large adoption of smart sensors. In this scenario, Machine Learning techniques are essential in leveraging the huge amount of gathered data to make intelligent decisions in many manufacturing tasks. Other techniques adopted include the more specific fields of Computer Vision and Deep Learning. With the adoption of Machine Learning, "manufacturers can gain insight to optimise the productivity of individual assets as well as the total manufacturing operation" [46].

There are several possible applications of ML to Industry 4.0. The main areas

are the following, according to [46]:

- **Inspection and monitoring**: these tasks are often performed with the adoption of computer vision techniques applied to both images and videos. RGB cameras are combined with ML-based algorithms to obtain high-throughput part inspection. Inspection and monitoring mostly consist of the detection of defects in a wide range of products. Images (or videos) could be subjected to some data pre-processing. After that, if Deep Learning is exploited, images are directly passed to a Neural Network model, that automatically performs the feature extraction stage and the following classification/detection. With a more classical approach, features engineering may be used, followed by a ML algorithm [46].
- **Fault detection**: for manufacturing companies, reducing machine down-times is vital to stay competitive by reducing costs and time waste. Also in this task, ML play an important role, allowing a timely and accurate diagnosis of manufacturing equipment process faults. Often, Deep Learning techniques, such as RNN or LSTM that can exploit time series data, but also classic ML techniques, are adopted [46].
- **Cloud manufacturing**: given the production of big data in Smart factories, cloud storage within manufacturing is acquiring more and more importance. ML algorithms can be used in cloud manufacturing to improve performance, reduce costs, and drive profitability [46].
- **Process improvement and optimisation**: manufacturing processes can be improved and optimised through ML. Indeed, the analytic capabilities of ML can be used to select the optimal set of parameters related to a given manufacturing process. Researchers state that this new discipline, linking ML with process improvement, will be subject to great growth in the coming decades [46].
- **Predictive maintenance (PdM)**: a new maintenance policy born with Industry 4.0. Data extracted from industrial equipment can be exploited to monitor its status and, thanks to the predictive capabilities of ML, increase its operational life and minimise machine downtime. A popular PdM task is the Remaining Useful Life estimation, that consists in predicting the remaining amount of operational life (in terms of cycles, days, hours...) of a given equipment.

1.2 Automated Machine Learning

1.2.1 What is AutoML?

In the last decade, Machine Learning acquired more and more importance in many different areas, even beyond the manufacturing field, like healthcare

industry, autonomous driving, natural language processing, and so on. This gain in popularity and the consequent growth in requests for ML solutions from companies has made figures like Data Scientist and Machine Learning engineer increasingly in demand. Indeed, the ML pipelines are characterized by several sensitive phases, like data cleaning or model selection, that require a lot of experience, domain knowledge and highly specialized data scientist. However, the lack of such experts and the high costs can be an obstacle and discourage the adoption of these technologies, especially for small and medium-sized companies, which would not have resources to create teams in this specific field. Moreover, there are several tasks, like hyper-parameter tuning, that are tedious and repetitive both for expert and non-experts. Experts may prefer to devote their time to more useful and motivating tasks, like interpreting the models' results or analyze the data [5] [59].

These are some reasons for which techniques for automating the ML procedures have begun to emerge in recent years, under the term of Automated Machine Learning (AutoML). The AutoML topic is being discussed in the literature, in particular in terms of the degrees of automation of the ML tasks. Researchers disagree on the role humans play in AutoML, and there exist two opposite perspectives: the first considers humans as part of the entire process, while the second aims at full automation. From this discussion, it is possible to categorise AutoML solutions into different types. With Narrow AutoML, experts remain part of the process and only some specific ML parts are handled automatically (usually, model selection and hyper-parameter optimization). On the opposite side, Generalised AutoML makes AutoML accessible to many people thanks to full automation, which also ideally removes the need for experts. The problem with this approach is that generally the obtained solutions are less transparent and the system less customisable. A third approach, known with the expression Human In The Loop (HITL), involves collaboration between humans and artificial intelligence. AutoML systems are monitored by humans who can support the difficulties of the system using intuition and domain knowledge. In general, the participation of humans makes the use of ML methods more socially acceptable [59].

Often, the AutoML tools focus on automating the model generation task. In this case the automated ML pipeline is described in Figure 1.3 from [5]. As visible, the model generation includes the learning algorithms used and the hyper-parameter optimization techniques. Note that Figure 1.3 does not

include preprocessing steps, e.g., data cleaning, or other tasks such as feature engineering and dimensionality reduction, that may, however, be tackled by AutoML tools [5].

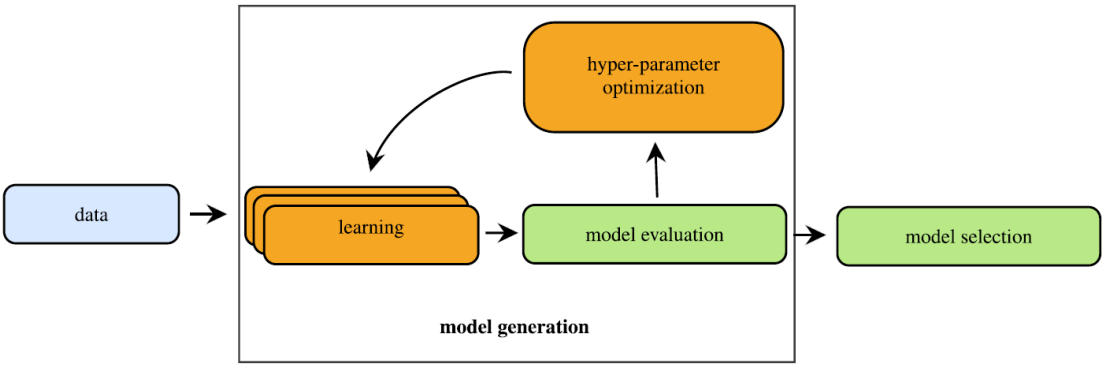


Figure 1.3: Overview of the automated machine learning pipeline for the CASH problem [5].

The model generation task is so important in Machine Learning that is also known by the acronym CASH, which stands for Combined Algorithm Selection and Hyper-parameter problem. Following the formalism introduced in [5], a mathematical formulation of the CASH problem can be provided. Let $\mathcal{A} = \{A^1, ..., A^a\}$ be a set of algorithms. Each algorithm A^i has a set of hyper-parameters in a domain Λ^i. So, with p hyper-parameters, the hyper-parameter space is defined as $\Lambda^i = \Lambda^i_1, ..., \Lambda^i_p$. Then, given a dataset X, the problem objective can be defined as the pair given by algorithm and hyper-parameter setting that minimizes the following loss:

$$A^*, \lambda^* \in \arg \min_{A^i \in \mathcal{A}, \lambda \in \Lambda^i} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}(A^i_\lambda, X^j_{train}, X^j_{valid}), \tag{1.1}$$

where $\mathcal{L}(A^i_\lambda, X^j_{train}, X^j_{valid})$ measures the loss obtained by algorithm A^i with hyper-parameters configuration λ on X^j_{valid} when trained on X^j_{train} with K Cross-Validation folds [5].

Clearly, the CASH problem can be addressed directly only when dealing with a supervised task, where the dataset X is provided together with ground truth labels describing, for example, the class to which each instance belongs. In this case, the model evaluation phase can be performed and models can be compared in terms of some quantitative metric. However, there are some tasks that need to be treated as unsupervised ones, especially in the industry field.

The problem in this last scenario cannot be tackled directly, and some solutions have been proposed in the literature.

1.2.2 Examples of AutoML tools

The interest in the topic of Automated Machine Learning started to increase about six years ago, as visible in Figure 1.4.

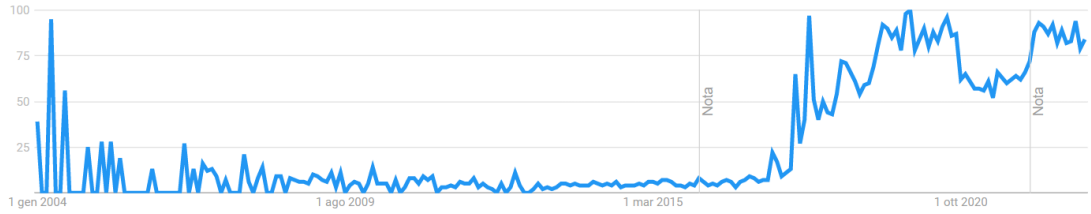


Figure 1.4: Google trend for the "Automated Machine Learning" topic.

During these years, several AutoML tools have been developed, with different levels of automation. Some of them are briefly described here.

- **TPOT** [39]: the Tree-based Pipeline Optimization Tool is one of the first AutoML tool and can be used for classification and regression tasks. It combines genetic programming and the scikit-learn library to provide the best models for a given dataset. Models are compared in terms of quantitative performance, so only supervised tasks are managed. TPOT provides support for Neural Networks, but data pre-processing is not automated [7] [59].
- **Auto-WEKA** [32]: it solves the CASH problem by Bayesian optimisation, but it does not support other steps, beyond working with supervised learning only [59] [5].
- **AutoSklearn** [19]: built around the sklearn library, it aims to solve the CASH problem. Similarly to Auto-WEKA, it employs Bayesian optimisation, but also the meta-learning technique to initialise the algorithm and hyper-parameter selection (warm-start problem) [59] [5].
- **AutoKeras** [30]: it was published in 2017, with the goal of finding autonomously neural network configurations using Bayesian optimisation. It is based on the well-known Deep Learning library Keras. Auto-Keras is built for supervised tasks and can handle images and text data [7] [5].
- **AutoGluon** [17] [45]: developed by Amazon, it uses both Machine Learning and Deep Learning algorithms to manage different applications. It can handle image, text, and tabular data and provides tools for object detection. AutoGluon Tabular combines multiple models to create ensambles [7].

All these tools can manage some ML tasks, while there exist other that can be considered in the framework of Generalised AutoML, such as Cloud AutoML by Google, Microsoft Azure Automated ML and Amazon SageMaker Autopilot. They are less transparent and customisable, but can be used also by non-experts since they require little programming [59].

1.2.3 AutoML challenges

When building an AutoML tool, one has to take into account a whole series of critical issues. Firstly, when dealing with the CASH problem, a search space for the algorithm hyper-parameters must be defined, taking care of not missing well-performing regions. In this context, the cold-start problem plays an important role. It is a problem where the process starts with a bad hyper-parameter configuration, or even with a bad model, consequently spending too much time to get better results. The meta-learning technique, explained in Subsection 2.2.1, can be used to deal with this problem. By looking at the similarities between the considered dataset and some historical ones, it is possible to warm-start the search problem beginning with better configurations and/or models.

AutoML tools could also be used by non-experts, or people with low ML knowledge. Alternatively, experts may use them to speed up their works, automating some tedious tasks. In both cases, the running time of the automated pipeline is a crucial aspect to take into account. An unexperienced user may prefer a suboptimal solution rather than having to wait too long for a solution. At the same time, for an expert, the advantage derived from automating ML tasks may not be as effective if waiting times are too long.

In the Machine Learning field there exist datasets of widely varying dimensionalites. If the AutoML tool has to deal with high-dimensional data that can affect the performance of the algorithms, a dimensionality reduction process should be integrated in the feature extraction/selection stage of the AutoML pipeline. Moreover, there are datasets with different sizes. The system must be able to scale on large datasets, but possibly also perform well with few data.

To evaluate the different models and/or hyper-parameter configurations, a performance metric (or more than one) must be carefully selected. This is a nontrivial problem, since the metric depends on several factors, among which there are the learning task and the type of data. For supervised tasks, the choices are numerous. The problem arises when dealing with an unsupervised task and

in the industrial scenario this is a common situation. For example, there exists no unsupervised performance metric to evaluate the unsupervised case of the well-known anomaly detection task [5].

The lack of transparency may be an obstacle to the adoption of AutoML technologies. Indeed, these tools are often seen as black boxes, since it is difficult to understand the reasons behind their decisions. This fact has a negative impact on the trust of the inexperienced user towards AutoML. To address the problem, explainability tools can be integrated in the system. In the "Human-in-the-loop" framework, experts can, among others, help the final user to understand the results of the AutoML system through explainability [59].

Since one of the goals of AutoML is to democratise Machine Learning by making it accessible to as many people as possible, even non-experts, another relevant thing to evaluate is the user experience quality. For example, in [7], some AutoML tools are compared in terms of documentation, simplicity of first use, and logs. Clearly, in this case only a qualitative evaluation can be done. The documentation analysis include considerations regarding the tutorials, the presence of examples of use-cases for beginners, and the level of detail of the documentation itself. Users might quickly abandon a tool if it is hard to initially configure. So, a good AutoML tool should be easy to setup requiring few lines of code to start; also, the simplicity of model export and import must be considered. The logs help developers to understand what is going on in the system. Good logs are essential for increasing transparency of AutoML tools, making them more accessible to different kind of users [7].

1.3 Introduction to Thesis Work

The work presented in this thesis is the result of an internship at Statwolf Data Science s.r.l. The work consisted in the development of AutoML solutions for some Machine Learning tasks related to Industry 4.0. In particular, two manufacturing tasks are addressed that are the Anomaly Detection, both with tabular and visual data, and the Remaining Useful Life (RUL) estimation. The first consists in the binary classification of unsupervised data into two unbalanced classes, one for normal instances and one for anomalous instances. The RUL estimation is a regression task in which the goal is to predict the remaining operational life of an industrial equipment.

The internship is part of the activities related to the "Automated Machine Learning for Advanced Monitoring in Digital Manufacturing and Industry 4.0" (AutoML 4.0) project. The aim of this project is the development of an AutoML tool to quickly enable data-driven Smart Monitoring for manufacturing companies. The Industry 4.0 is gaining popularity during recent years, but many companies, and in particular the SMEs, continue to face some difficulties in exploiting the potential of data gathered from industrial processes and equipment. The main problem remains the lack of qualified employees, e.g., Data Scientists and Machine Learning engineers, and for this reason an automated tool for developing ML solutions would be very valuable. This AutoML tool is designed to also be used by people with little or no ML experience, and could also motivate companies to invest more in data-driven solutions. Compared to the categorisation presented in Subsection 1.2.1, AutoML 4.0 falls within the Human-in-the-loop framework, since it is a decision support system in which human experts act as supervisors.

AutoML 4.0 is one of the activities of a broader project, the EUHubs4Data (EUH4D), founded by the European Union with the objective of building a European federation of Data Innovation Hubs, where data sources and data-driven services and solutions will be made accessible to European SMEs, start-ups, and web entrepreneurs. The ultimate goal is to reduce the lag behind in data-driven innovation that characterises most of Europe's SMEs.

The StatwolfML library already provides an AutoML tool able to manage both classification and regression tasks for tabular data. In particular, after inferring the type of task, it automatically performs some pre-processing steps, like handling the Nan values and the outliers. Then, it finds the best hyper-parameter configuration for the provided dataset, returning the corrisponding model. However, the manufacturing tasks considered during the internship are somehow peculiar, such as the AD one, which is not supervised. So, handle these tasks require specific modules being able to managing all related issues.

In the thesis, the work on Anomaly Detection on tabular data can be found in Chapter 2, while Chapter 3 is dedicated to the Visual AD. After a brief introduction to Predictive Maintenance, Chapter 4 focus on the RUL estimation task, while the final Chapter 5 recaps the main results and the future work directions.

2

Anomaly Detection on Tabular Data

2.1 Introduction

Anomaly Detection (AD) is a task that consists in the identification of rare items, events or observations, that deviate significantly from the majority of the points in a dataset. In other words, given a well-defined notion of normal behavior, the anomalous instances do not conform to it. Since this kind of points can be thought as outliers with respect to the distribution of the normal ones, the task is also called Outlier Detection (OD). In this chapter, the data considered for the AD task are in tabular format, where rows represent the single instances that can be classified as normal or not, while columns refer to the features describing the instances.

Anomaly Detection can be treated as a supervised or unsupervised task. However, the lack of annotated data makes it often difficult to deal with supervised AD. Especially at the beginning of a digital transition, for many companies, annotating data is a costly activity, both in terms of time and for the need of engaging domain experts. For this reason, the AD task is often addressed in its unsupervised version, more attractive to companies because of its cost-effectiveness and quicker results. Possibly, in the future, increased awareness among companies of the potential of AD could prompt them to invest in order to collect labelled data. However, in this chapter the focus is on unsupervised AD.

The absence, in unsupervised AD, of hold-out data with labels combines

with a second problem, that is, the lack of a universal objective function to guide model selection. Indeed, the definition of anomaly may differ depending on the considered domain and consequently also on the objective function to be optimised. These two critical issues contribute to making the model selection for the AD task a "black art" [64], since model evaluation/comparison is not feasible. In general, there are no principled work on model selection for unsupervised OD [64].

An AutoML module for the Outlier Detection task, that can be called AutoOD, should be able to automatically select a good outlier method and its hyperparameters, i.e., it should autonomously perform the model selection task. Given all the issues previously highlighted, this goal is very challenging for unsupervised OD. In the literature, some solutions have been proposed: the state-of-the-art according to [5] is visible in the table of Figure 2.1, which includes the most recent approaches. Notice that, according to the column "Evaluation" of Figure 2.1, there are only two methods that satisfy the constraint of dealing with an unsupervised task, that are MetaOD [64] and Meta-AAD [63], both leveraging the meta-learning technique.

Method	Search	Evaluation	Incremental	Pros	Cons
PyODDS [48]	Global optimization	Supervised	No	It is scalable and contains several AD algorithms	It does not address cold-start issue
LSCP [84]	Grid search	Supervised	No	It uses a similarity measure to assign ground truth labels	It is composed of only one base model where the performance may degrade when data have irrelevant attributes. It is also costly due to the neighbors' search
TODS [45]	Grid search	Supervised	No	It includes several components and detectors to the pipeline	The used search technique is not mentioned. It also uses a supervised evaluation measure
MetaOD [86]	–	Unsupervised	No	It handles the cold-start issue using meta-learning and meta-features	It discretizes the parameter space and is not a fully supervised method
Meta-AAD [83]	–	Unsupervised	Yes	It performs model selection using reinforcement learning and is applied to any unlabeled data. It is efficient since it uses only 6 features instead of the whole feature space	Many features are ignored and so less information is available
AutoOD [46]	RNN	Supervised	No	It is also built using reinforcement learning for tuning	It works with deep anomaly detection algorithms which are costly, and has been evaluated using ground truth labels

Figure 2.1: State-of-the-art approaches for AutoOD according to [5].

The other approaches, namely PyODDS [37], LSCP [65], TODS [34] and

AutoOD [36], despite using unsupervised algorithms, compare different models and hyper-parameter configurations by means of supervised metrics, often the F1-score, which, however, require labelled data. Between MetaOD and Meta-AAD, the most promising method seems to be the first, since considers a much larger set of meta-features. Moreover, the second method adopts reinforcement learning, so requiring more time in training the meta-learner.

The working principle of MetaOD is detailed described in Section 2.2, while Section 2.3 includes some experiments with this method, that is compared with the Isolation Forest algorithm. Finally, Section 2.4 deals with the intepretability topic for the AD task.

2.2 MetaOD

2.2.1 Meta-learning

In principle, meta-learning is a Machine Learning field composed by a suite of techniques that exploit past experience on a set of prior tasks to perform efficient learning on a new task [64]. In particular, as described in Figure 2.2, several models with various hyper-parameter configurations λ_i are trained and evaluated on different datasets. This phase is known as the meta-learner training. The results in a Performance matrix P, whose elements $P_{i,j}$ are the performance of the hyper-parameters configuration i on the training data j. Then, from each training data j, several features m_j, called meta-features, that describes characteristics of the data, are collected. These meta-features are used to assess similarities between the new data and the ones used to train the meta-learner. So, given a new dataset, it is possible to identify the most similar datasets and provide as output the configurations that perform better on these, which, hopefully, are the best ones also for the new data.

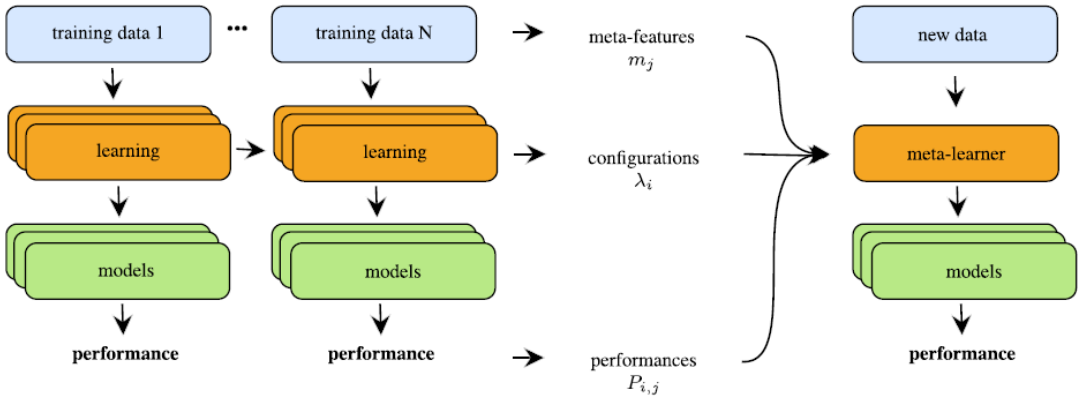


Figure 2.2: Meta-learning technique schema from [5].

Clearly, meta-learning is based on the assumption that datasets are comparable in terms of some characteristic features, whose choice is critical for the proper functioning of the technique. Meta-features that can be extracted from a dataset are of two types [64]:

1. **Statistical features**: features that capture statistical properties of the underlying data distributions. E.g, min, max, mean, median, correlation, covariance, etc. of features and features combinations.
2. **Landmarker features**: problem-specific features. In the case of MetaOD, the OD models iForest, HBOS, LODA and PCA are applied on the datasets and meta-features are extracted from the models' structure and from the outlier scores given in output by the models. E.g., average horizontal and vertical tree imbalance from the iForest algorithm.

Although statistical features are commonly used in meta-learning, the optimal set of meta-features has been shown to be application dependent [57]. In Figure 2.3 there is a summary of the statistical and landmark features adopted by MetaOD.

2.2.2 MetaOD Problem Statement

MetaOD is the first meta-learning approach to OD, that selects an effective model (that is, detector and hyperparameter configuration) to be employed in a new detection task [64]. Following the meta-learning paradigm, a new OD dataset, without labels, is compared to some historical datasets, with labels, coming from a meta-train database. To assess the performance of a given model on this new dataset, MetaOD looks at its prior performance on similar datasets. In this way, this approach is able to handle the unsupervised version of OD,

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability	$\frac{p}{n}$, $\log(n)$, $\log(\frac{n}{p})$
Nr features	p	Curse of dimensionality	$\log(p)$, % categorical
Sample mean	μ	Concentration	
Sample median	$\tilde{X}$	Concentration	
Sample var	σ^2	Dispersion	
Sample min	$\max_X$	Data range	
Sample max	$\min_X$	Data range	
Sample std	σ	Dispersion	
Percentile	P_i	Dispersion	q1, q25, q75, q99
Interquartile Range (IQR)	$q75 - q25$	Dispersion	
Normalized mean	$\frac{\mu}{\max_X}$	Data range	
Normalized median	$\frac{\tilde{X}}{\max_X}$	Data range	
Sample range	$\max_X - \min_X$	Data range	
Sample Gini		Dispersion	
Median absolute deviation	$\text{median}(X - \tilde{X})$	Variability and dispersion	
Average absolute deviation	$\text{avg}(X - \tilde{X})$	Variability and dispersion	
Quantile Coefficient Dispersion	$\frac{(q75-q25)}{(q75+q25)}$	Dispersion	
Coefficient of variance		Dispersion	
Outlier outside 1 & 99	% samples outside 1% or 99%	Basic outlying patterns	
Outlier 3 STD	% samples outside 3σ	Basic outlying patterns	
Normal test	If a sample differs from a normal dist.	Feature normality	
kth moments			5th to 10th moments
Skewness	Feature skewness	Feature normality	min, max, μ, σ, skewness, kurtosis
Kurtosis	$\frac{\mu_4}{\sigma^4}$	Feature normality	min, max, μ, σ, skewness, kurtosis
Correlation	ρ	Feature interdependence	min, max, μ, σ, skewness, kurtosis
Covariance	Cov	Feature interdependence	min, max, μ, σ, skewness, kurtosis
Sparsity	$\frac{\#\text{Unique values}}{n}$	Degree of discreteness	min, max, μ, σ, skewness, kurtosis
ANOVA p-value	p_{ANOVA}	Feature redundancy	min, max, μ, σ, skewness, kurtosis
Coeff of variation	$\frac{\sigma_x}{\mu_x}$	Dispersion	
Norm. entropy	$\frac{H(X)}{log_2 n}$	Feature informativeness	min,max, σ, μ
Landmarker (HBOS)	See §B.2	Outlying patterns	Histogram density
Landmarker (LODA)	See §B.2	Outlying patterns	Histogram density
Landmarker (PCA)	See §B.2	Outlying patterns	Explained variance ratio, singular values
Landmarker (iForest)	See §B.2	Outlying patterns	# of leaves, tree depth, feature importance

Figure 2.3: Selected meta-features for characterizing an arbitrary dataset [64].

provided that you have enough OD supervised datasets to properly train the meta-learner.

MetaOD includes a phase of training of the so-called meta-learner. Following the description in [64], this phase involves:

- A meta-train database, i.e., a collection of historical OD datasets, $D_{train} = \{D_1, ..., D_n\}$, provided with ground truth labels, where $D_i = (X_i, y_i)$, for $i = 1, .., n$.
- Performances of the m models, that defined the model space $\mathcal{M}$, on the meta-train datasets.

So, given a new input dataset X_{test} with no labels, the OD model selection problem consists in select a model $m \in \mathcal{M}$ to employ in the new test task [64].

The historical datasets are 62 independent datasets coming from three sources:

1. 23 datasets from the ODDS library.
2. 19 DAMI datasets.
3. 20 benchmark datasets [16].

A MetaOD model is defined as a pair (detector, configuration) for a specific hyper-parameter configuration, where the hyper-parameter space is discretized. In Figure 2.4 all the models considered are summarised. For instance, a model can be the detector iForest combine with the configuration $n\_estimators = 100$ and $max\_features = 0.1$.

Detection algorithm	Hyperparameter 1	Hyperparameter 2	Total
LOF (Breunig et al., 2000)	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	distance: ['manhattan', 'euclidean', 'minkowski']	36
kNN (Ramaswamy et al., 2000)	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['largest', 'mean', 'median']	36
OCSVM (Schölkopf et al., 2001)	nu (train error tol): [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	kernel: ['linear', 'poly', 'rbf', 'sigmoid']	36
COF (Tang et al., 2002)	n_neighbors: [3, 5, 10, 15, 20, 25, 50]	N/A	7
ABOD (Kriegel et al., 2008)	n_neighbors: [3, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	N/A	7
iForest (Liu et al., 2008)	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	max_features: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	81
HBOS (Goldstein & Dengel, 2012)	n_histograms: [5, 10, 20, 30, 40, 50, 75, 100]	tolerance: [0.1, 0.2, 0.3, 0.4, 0.5]	40
LODA (Pevný, 2016)	n_bins: [10, 20, 30, 40, 50, 75, 100, 150, 200]	n_random_cuts: [5, 10, 15, 20, 25, 30]	54
			302

Figure 2.4: Outlier Detection models adopted in MetaOD [64].

2.2.3 MetaOD Working Principle

MetaOD work is characterised by two phases, described in the schema of Figure 2.5:

1. **Offline Meta-Learner Training**: training of the meta-learner with the meta-train database.
2. **Online Model Selection**: the best models are predicted for a new unsupervised dataset.

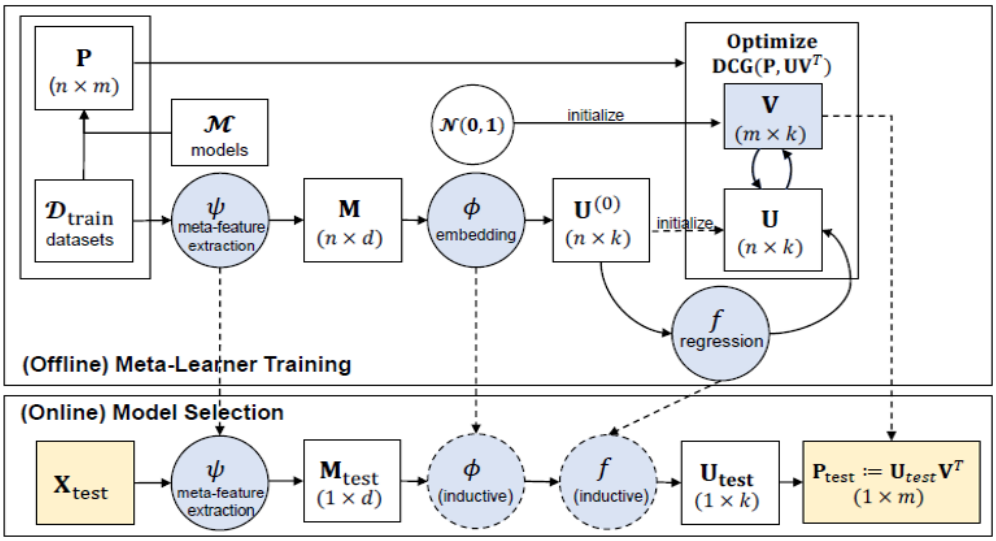


Figure 2.5: MetaOD overview; components that transfer from offline to online (model selection) phase are shown in blue [64].

In the first step of the Offline phase (circled in red in Figure 2.6), MetaOD constructs the performance matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$ by running and evaluating all m models of the model space $\mathcal{M}$ on all n meta-train datasets. On the other hand, to capture task similarity, $d = 200$ meta-features are extracted from each meta-train dataset, obtaining the matrix $\mathbf{M} = \psi(\{X_1, ..., X_n\}) \in \mathbb{R}^{n \times d}$, where ψ is the meta-features extractor.

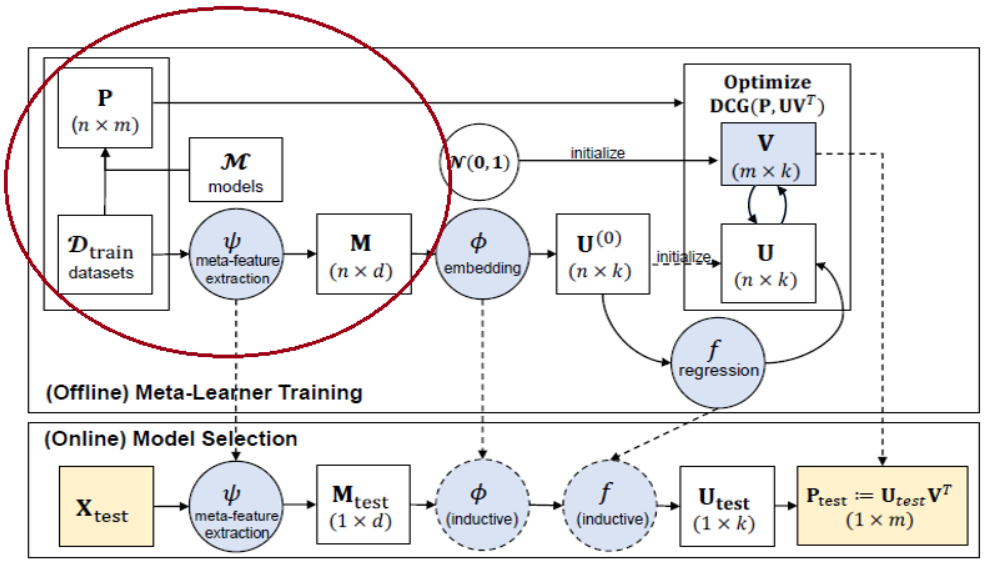


Figure 2.6: MetaOD overview with the computation of matrices **P** and **M** circled in red[64].

The training of the meta-learner consists of a factorization of the matrix **P** into matrices $\mathbf{V} \in \mathbb{R}^{m \times k}$ and $\mathbf{U} \in \mathbb{R}^{n \times k}$. Since the goal is to rank the models for each dataset row-wise [64], MetaOD uses a rank-based criterion called Discounted Comulative Gain (DCG). Indeed, DCG allows ranking the models according to their performance on the datasets, that is not possible when the factorization is obtained, for instance, by minimizing . Indeed, with classical factorization methods, like by minimizing the Frobenius norm of $\mathbf{P} - \mathbf{U}\mathbf{V}^T$, it is not possible to sort the models on the base of their performance on the datasets.

The DCG for a dataset i is defined as:

$$DCG_i = \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\log_2 \left(1 + \sum_{k=1}^{m} \mathbf{1}\left[\hat{\mathbf{P}}_{ij} \leq \hat{\mathbf{P}}_{ik}\right]\right)}, \tag{2.1}$$

where $\hat{\mathbf{P}}_{ij} = \langle \mathbf{U}_i, \mathbf{V}_j \rangle$ is the predicted performance of model j on dataset i, while b is a scalar.

Since DCG is not differentiable, the indicator function is replaced by the smooth sigmoid approximation:

$$DCG_i \approx sDCG_i = \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\log_2 \left(1 + \sum_{k=1}^{m} \sigma \left(\hat{\mathbf{P}}_{ij} \leq \hat{\mathbf{P}}_{ik}\right)\right)}. \tag{2.2}$$

So, the training step, highlighted in Figure 2.7, consists of optimising, through Stochastic Gradient Descent (SGD), the following smoothed objective function:

$$\mathcal{L} = -\sum_{i} sDCG_i(\mathbf{P}_i, \mathbf{U}_i\mathbf{V}^T). \tag{2.3}$$

Optimisation through SGD is obtained by alternately solving $\mathbf{U}$ after fixing $\mathbf{V}$ and vice versa.

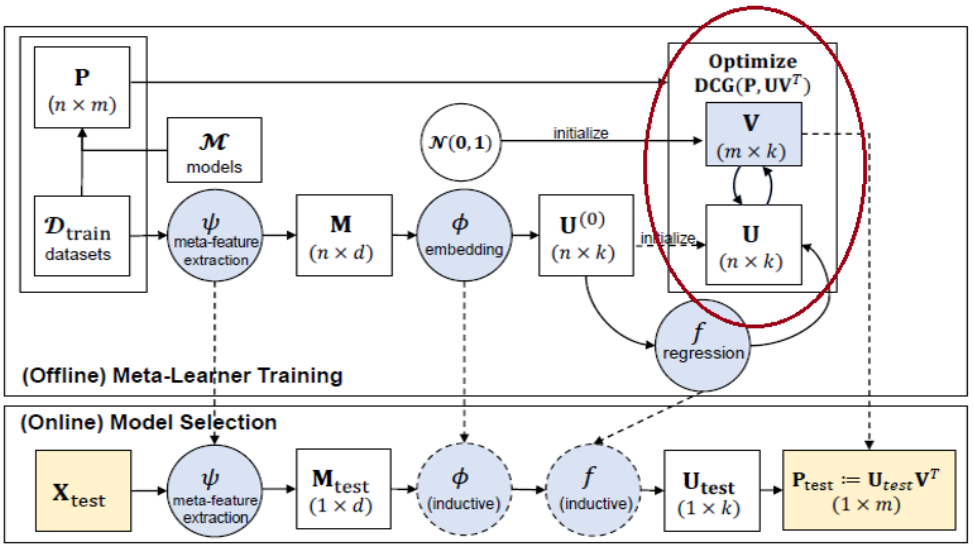


Figure 2.7: MetaOD overview with the DCG factorization of matrix $\mathbf{P}$ circled in red[64].

Initialisation plays an important role in non-convex problems [64]. For the training of the meta-learner, it is in particular important to initialize properly the matrices $\mathbf{U}$ and $\mathbf{V}$ (Figure 2.8):

- $\mathbf{U}^{(0)} = \phi(\mathbf{M})$, where ϕ is a dimensionality reduction function (i.e., a PCA), $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^k$, $k < d$, applied to the meta-features matrix $\mathbf{M}$.
- $\mathbf{V}^{(0)} = \mathcal{N}(0, 1)$

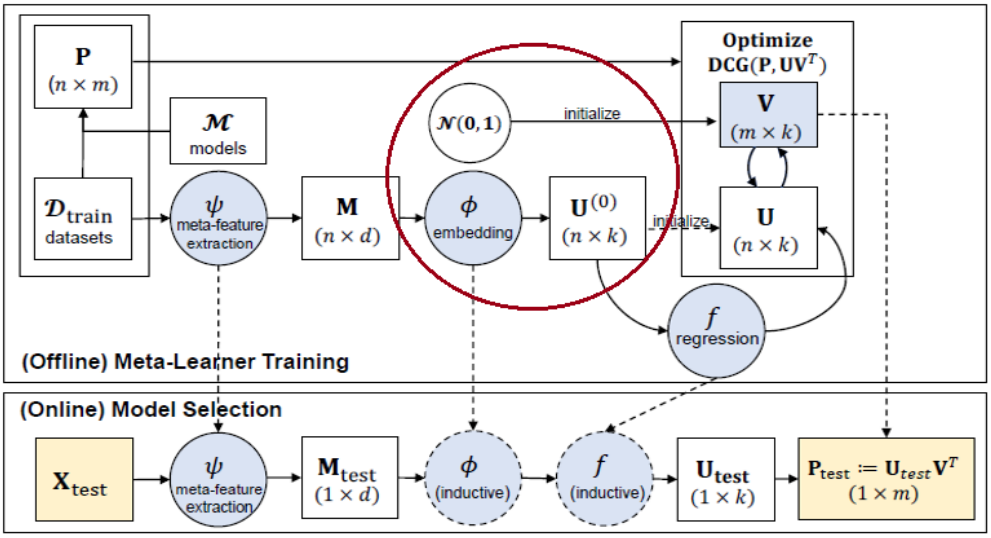


Figure 2.8: MetaOD overview with the initializations of matrix **U** and **V** circled in red[64].

The last ingredient of the offline meta-learner training is highlighted in Figure 2.9. MetaOD learns also a random forest regressor $f : \mathbb{R}^k \rightarrow \mathbb{R}^k$ that maps the initial version of matrix **U**, $\mathbf{U}^{(0)}$, that includes the embedding features $\phi(\mathbf{M})$, onto the final optimized **U**, obtained at the end of the training phase. The utility of such a regressor is more clear in the Model Selection phase.

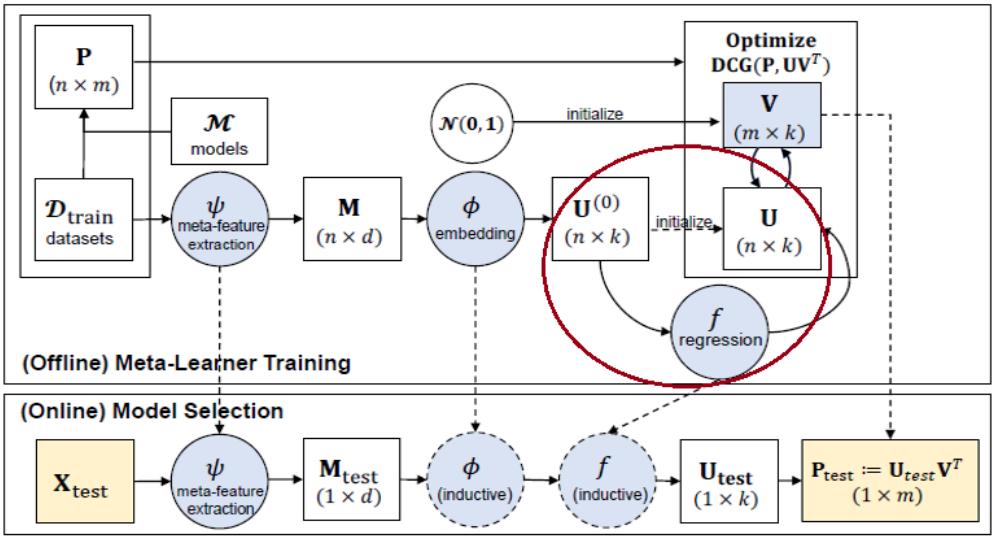


Figure 2.9: MetaOD overview with the random forest regressor circled in red[64].

The second phase of the MetaOD working, that is, the Online Model Selection, is more linear. Given a new unsupervised dataset X_{test}, it undergoes the

following processes:

1. Computation of the meta-features $\mathbf{M}_{test} = \psi(x_{test}) \in \mathbb{R}^d$.
2. Embedding of meta-features through PCA, $\phi(\mathbf{M}_{test}) \in \mathbb{R}^k$.
3. The embeddings are passed to the random forest regressor f, to obtain matrix $\mathbf{U}_{test} = f(\phi(\mathbf{M}_{test})) \in \mathbb{R}^k$
4. Finally, the performance of each model is obtained as $\mathbf{P}_{test} = \mathbf{U}_{test}\mathbf{V}^T \in \mathbb{R}^m$.

Basically, MetaOD returns a vector, $\mathbf{P}_{test}$, that contains the performance of each model on the new data X_{test}. The solution to the MetaOD Model Selection problem consists of selecting the model with the best performance:

$$\arg\max_j \langle f(\phi(\psi(\mathbf{X}_{test})), \mathbf{V}_j \rangle. \tag{2.4}$$

2.3 Experiments with MetaOD

2.3.1 Datasets

MetaOD is tested on two unsupervised OD datasets, taken form two different industrial scenarios.

The first dataset is the DATACENTER MONITORING DATA provided by CINECA, that is involved in the EUH4D project. It describes the jobs runned on Marconi100 system in the period October 2020 - February 2021. The raw data are stored in two kind of files, the .jobs and the SACCT ones, whose tabular structure is described in the examples of Figures 2.10 and 2.11, respectively.

```
job_data.head()
```

	data	jobID	wcl	nproc_assigned	cpu_aff	mem	queue	exectime	queuetime	status	acc	user	nmic	ngpus	QOS	nproc_required
0	20201001	771740	86400	32	1	243200	m100_usr_prod	26460	20878	0:0	0	4	normal	32	NaN	NaN
1	20201001	771741	86400	32	1	243200	m100_usr_prod	26294	120	0:0	0	4	normal	32	NaN	NaN
2	20201001	772027	86400	32	1	243200	m100_usr_prod	26734	20684	0:0	0	4	normal	32	NaN	NaN
3	20201001	772028	86400	32	1	243200	m100_usr_prod	26357	101	0:0	0	4	normal	32	NaN	NaN
4	20201001	772198	86400	32	1	243200	m100_usr_prod	24264	10450	0:0	0	4	normal	32	NaN	NaN

Figure 2.10: Structure of data in .job files.

```
sacct_data.head()
```

	jobid	partition	qos	reqnodes	allocnodes	reqcpus	alloccpus	submit	eligible	start	end	state	reqgres	allocgres	reqmem
0	23126	m100_usr_prod	normal	4	4	512	128	20200430-18:00:46	20200430-18:00:46	20200430-18:00:52	20210226-09:45:48	RUNNING	gpu:0	gpu:16	230Gn
1	24340	m100_usr_prod	normal	1	1	32	32	20200430-19:34:21	20200430-19:34:21	20200430-19:34:27	20210226-09:45:48	RUNNING	gpu:0	gpu:4	246000Mn
2	24341	m100_usr_prod	normal	1	1	32	32	20200430-19:34:21	20200430-19:34:21	20200430-19:34:27	20210226-09:45:48	RUNNING	gpu:0	gpu:4	246000Mn
3	24342	m100_usr_prod	normal	1	1	32	32	20200430-19:34:21	20200430-19:34:21	20200430-19:34:28	20210226-09:45:48	RUNNING	gpu:0	gpu:4	246000Mn
4	24343	m100_usr_prod	normal	1	1	32	32	20200430-19:34:21	20200430-19:34:21	20200430-19:34:28	20210226-09:45:48	RUNNING	gpu:0	gpu:4	246000Mn

Figure 2.11: Structure of data in SACCT files.

Each row describes a different job and most jobs are shared between the two kind of files. Only data in SACCT files are considered for the MetaOD evaluation, since the information provided for each job is more useful and detailed, in particular for the provided timestamps (columns "submit", "eligible", "start", and "end").

Data are aggregated by time slot of one hour: in this way, the anomaly detection problem consists of predicting if the hours are abnormal or normal. Secondarily, for each time slot, 27 features are computed:

- For each status of the job, the number of jobs ended up in the status in the time slot (9 features).
- For each status of the job, the percentage of jobs that ended in the status in the time slot (9 features).
- Mean and standard deviation of the execution times of the jobs terminated in the time slot (2 features).
- Mean and standard deviation of the queue times of the jobs terminated in the time slot (2 features).
- Ratio between the mean execution time and the mean queue time (1 feature).
- Mean difference between CPUs requested by the user and CPUs allocated by the system (1 features).
- The day time slot of the hour, considering slots of 8 hours (3 features).

The final dataset consists of 2967 rows.

Pratically, the dataset is only weakly supervised, since, together with the data, a list of incidents is provided, with some broad information on the data, the duration, and the cause, as shown in Figure 2.12.

	host	data	duration	why	what_recode
0	M100	26-gen-21	from 8:00 up to 13:40	change	Scheduled maintenance
1	M100	10-gen-21	from 11:30 to 13:00	incident	Filesystem issue
2	M100	23-dic-20	from 22 dec 2020 evening (h 19:00) to 12:20	incident	Login issue
3	M100	23-dic-20	from 22 dec 2020 evening (h19:00) to 14:40	incident	GPFS/login issue
4	M100	03-dic-20	from 10:00 up to 12:00	incident	Other
5	M100	02-dic-20	from 2/12 not solved	incident	Other
6	M100	01-dic-20	from 8:00 up to 12:45	change	scheduled maintenance
7	M100	06-nov-20	at 10:27	change	Other
8	M100	29-ott-20	at 16:00	change	Other
9	M100	21-ott-20	from 9:00 up to 14:00	incident	Other
10	M100	14-ott-20	from 13/10 up to 14/10	incident	Other
11	M100	13-ott-20	from 8:00 up to 20:00	change	Scheduled maintenance

Figure 2.12: Annotated incidents.

The second dataset comes from the Statwolf's partner Galdi Srl and it is called Packaging Industry Anomaly Detection (PIADE).

It includes data from five industrial packaging machines:

- Machine s_1: from 2020-01-01 14:00:00 to 2021-12-31 13:00:00.
- Machine s_2: from 2020-06-17 08:00:00 to 2021-12-31 07:00:00.
- Machine s_3: from 2020-10-07 12:00:00 to 2022-01-01 23:00:00.
- Machine s_4: from 2020-01-01 01:00:00 to 2022-01-01 23:00:00.
- Machine s_5: from 2020-01-20 08:00:00 to 2022-01-01 12:00:00

Also in this case, the raw data provided by the equipment are aggregated in time slots of one hour (for each different machine). The features computed for each slot are the following:

- Equipment_ID, that is the machine identifier.
- The number of changes in the machine state.
- The percentage of time spent in the downtime state.
- The percentage of time spent in the idle state.
- The percentage of time spent in the performance loss state.
- The percentage of time spent in the production state.
- The percentage of time spent in the scheduled downtime state.

- The sum of all alarm occurrences.
- Counters of the occurrences of each of the 133 different types of alerts.
- The numbers of transitions from one state to another.

In the evalaution of MetaOD, only data relative to Machine s_1 are considered, for a total of 8973 rows.

2.3.2 Isolation Forest

Given the unsupervised nature of the task, a possible way to evaluate the quality of the MetaOD's results consists in comparing them with the predictions of a well-established outlier detector, that is the Isolation Forest, or iForest (IF) [38]. IF operates by isolating anomalous points in a dataset, given two important assumptions about them:

1. They are very rare, so representing the minority.
2. The features' values of abnormal instances deviate a lot from the distribution of the features' values of the normal instances.

So, anomalies are few and they differ clearly from normal points, thus being easier to isolate.

Similarly to Random Forest, IF is an ensemble method and its strength comes from the combination of several weak detectors, called Isolation Trees, or iTrees. Each iTree induces a partitioning in the instances' space by iteratively selecting a random feature from the feature set and a random split point from the range of the feature's values. This process continues until either each instance is isolated in a leaf, or the tree reaches a pre-defined height limit. The intuition is that, since anomalies can be easily isolated, they generally require less partitions, so falling in leafs that are closer to the tree root. This can be better understood with the example of Figure 2.13, taken from the original paper. Point x_0 is considered an anomaly, while x_i is normal, as can be guessed from the distribution of points. As visible, isolating x_i is much harder than isolating x_0, requiring more partitions and therefore higher levels of the tree.

While the training consists in building such trees (whose number is an hyperparameter), in the test phase it is possible to derive an anomaly score for a candidate point from the average path length of the point in a collection of iTrees, the Isolation Forest. The path length of a point in a tree is defined as the number of edges in the path from the tree's root to the leaf in which the point

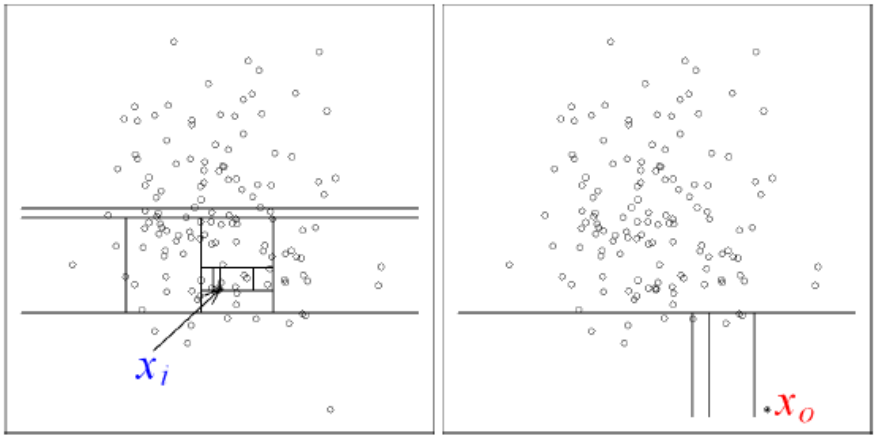


Figure 2.13: Example of isolation of a normal point, x_i, and an abnormal point, x_0 [38].

is isolated. The score, as defined in the paper, is between 0 and 1: if the value is close to 1 we have an anomaly, while close to 0 the instance is classified as normal.

It is important to note that IF performs random subsampling of training data, since it works better when the data sampling size is kept small [38].

Finally, the time complexity is linear with a low constant and a low memory requirement [38].

2.3.3 Results

The implementation of MetaOD adopted is the official one, from the authors of the paper. Note that all the included models come from the PyOD library. MetaOD results are compared to the ones of the Random Forest implementation provided by the StatwolfML library, with $n\_estimators = 100$ and $max\_features = 1.0$, where $max\_features$ specifies the portion of features involved in the training of each estimator. Another parameter to be specified is the $contamination$, that is the proportion of outliers in the dataset: it is set to 0.1 by default in all the MetaOD models.

Once the data are provided to MetaOD, it returns the best n models, where n can be specified by the user. For the CINECA Dataset, the first six best combinations of detector and hyper-parameters configuration are shown in Figure 2.14.

The Lightweight Online Detector of Anomalies (LODA) algorithm [44] with $n\_bins = 5$ and $n\_random\_cuts = 20$, is the best model for MetaOD. In Fig-

```
Showing the top recommended models...
0 LODA (5, 20)
1 OCSVM (0.1, 'poly')
2 OCSVM (0.2, 'poly')
3 OCSVM (0.1, 'sigmoid')
4 ABOD 3
5 OCSVM (0.3, 'poly')
```

Figure 2.14: Best models for the CINECA Dataset according to MetaOD.

ure 2.15 the anomalies detected by the IF (in green) are compared with the one of LODA (in purple). The contamination is set to 0.1 for the IF as well. The anomaly scores provided in the output of the two models are normalised between 0 and 1 and visualised with respect to the date. Since the dataset is weakly supervised, it is possible to compare results with the incidents annotated by CINECA. The beginning of an incident is reported by a vertical dotted red line, whereas its end is reported by a vertical dotted blue line.

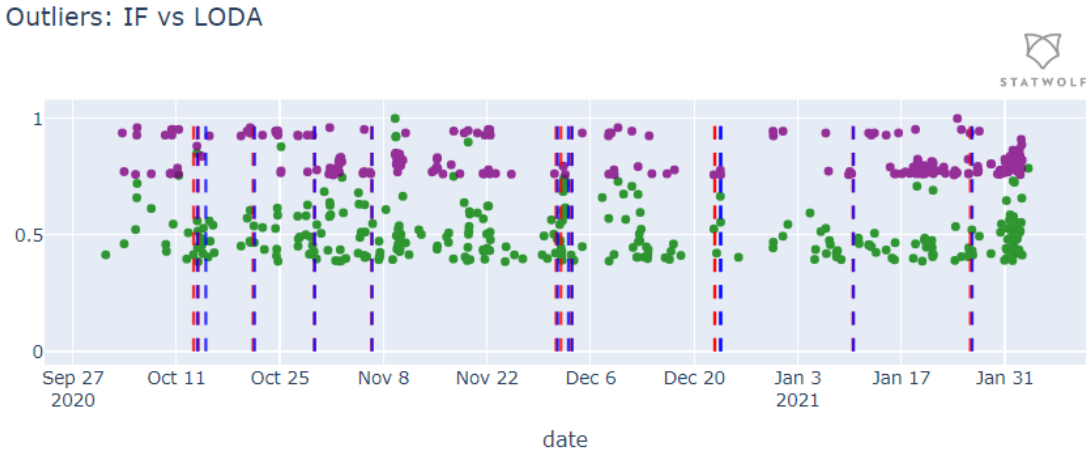


Figure 2.15: Comparison of detected anomalies by LODA (in purple) and by IF (in green), with normalized anomaly scores. The vertical dotted lines highlight incidents annotated by CINECA: red lines identify the beginning and blue lines the ending of the event.

Sometimes, there are matches between predictions and annotations, for instance in October 13^{th} and 14^{th}, December 1^{st}, 2^{nd} and 23^{rd}, while in other cases the anomalies seem anticipate the incident (October 21^{st}, January 10^{th}). However, there are also some mismatches, and, moreover, a lot of apparent false positives, i.e., detected anomalies with no correlated incidents. Unfortunately, the absence of more precise and complete annotations makes it difficult to evaluate performance.

Given the high number of detected anomalies, a post-processing is applied to the results, in order to allowing a better comparison between LODA and IF. In particular, an anomaly is kept only if at least three anomalies are observed before it in a window of 5 time slots. Figure 2.16 shows the result.

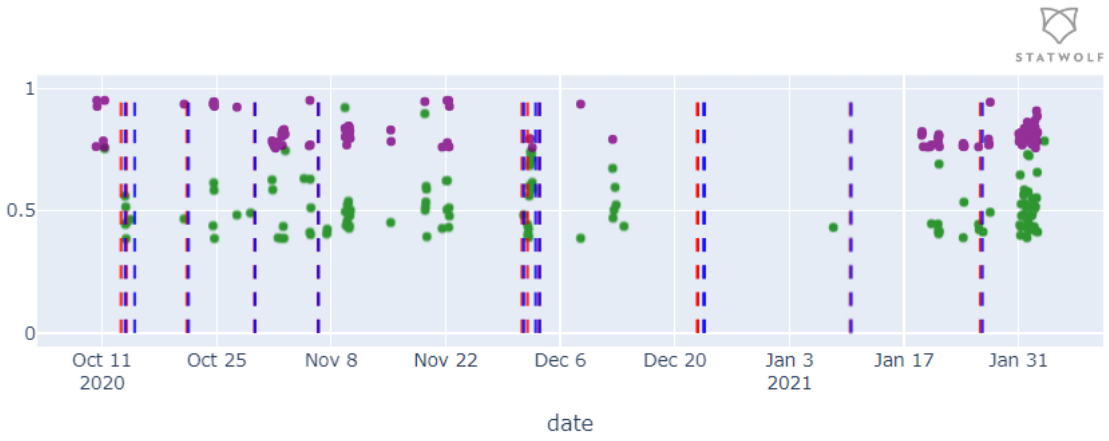


Figure 2.16: Comparison of detected anomalies by LODA (in purple) and by IF (in green), with normalized anomaly scores, after the post-processing. The vertical dotted lines highlight incidents annotated by CINECA: red lines identify the beginning and blue lines the ending of the event.

As can be seen, most of the anomalies captured by IF are also identified by LODA, so the two models show comparable results. The most evident differences are around October 11^{th}, where for LODA there is a cluster of anomalies, not identified by IF, and around October 13^{th}, when the opposite situation occurs.

The best models for MetaOD on the PIADE Dataset, instead, are summarized in Figure 2.17.

```
Showing the top recommended models...
0 ABOD 15
1 ABOD 20
2 ABOD 10
3 Iforest (150, 0.6)
4 ABOD 50
5 Iforest (20, 0.8)
```

Figure 2.17: Best models for the PIADE Dataset according to MetaOD.

The Angle-Based Outlier Detector (ABOD) [33] with $n\_neighbors = 15$ is the model with the best performance. Its predicted anomalies (in purple) are

compared to the one of IF (in green) in Figure 2.18. Given the larger dataset size, the contamination is set to 0.05 for both models. In this case, no annotations about incidents are provided, so the only possible considerations regard the comparison between the two models' predictions. Also in this case, the high number of detected anomalies can be reduced with a proper post-processing. With the same procedure used for the CINECA Dataset, it is possible to obtain the result of Figure 2.19. The predictions of IF and ABOD mostly match, in particular in correspondence of clustered anomalies, like in April/May 2020, March 2021, April 2021, and so on.

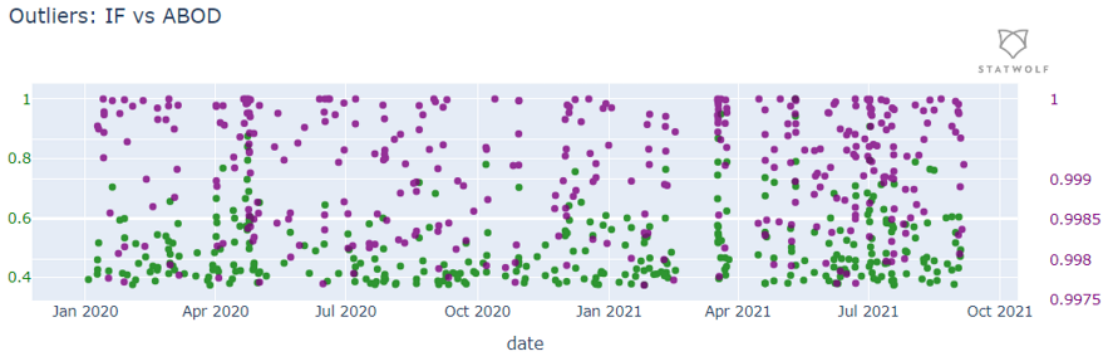


Figure 2.18: Comparison of detected anomalies by ABOD (in purple) and by IF (in green), with normalized anomaly scores.

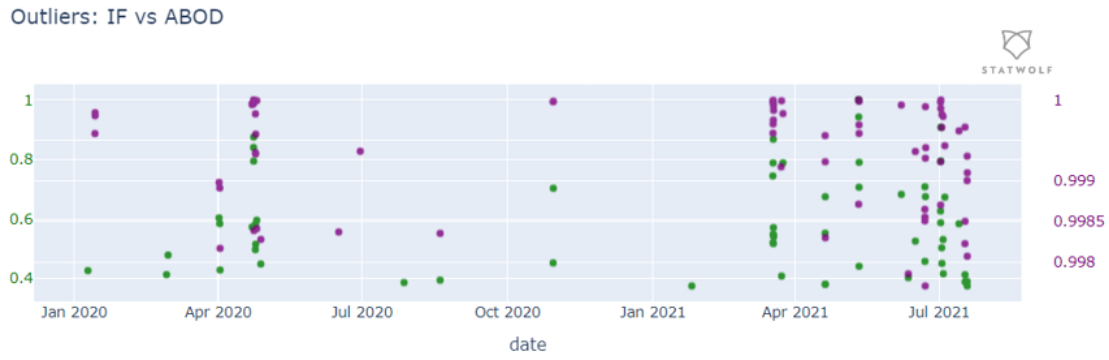


Figure 2.19: Comparison of detected anomalies by ABOD (in purple) and by IF (in green), with normalized anomaly scores, after the post-processing.

2.3.4 Experiment with a Synthetic Supervised Dataset

The last MetaOD evaluation is performed on a supervised dataset, created from scratch to make possible a quantitative evaluation of the approach.

The supervised dataset is obtained by sampling from two multivariate normal distribution, defined with the Numpy library. For the first distribution, the mean vector is a 30-dimensional vector of zeros, while the covariance matrix is the 30x30 identity. It is used to sample the 2700 normal points. The outliers (300) are instead drowned from a distribution with a mean vector that differs from the previous in the last 5 components, that have values 10, 12, 14, 16, 18. The 30x30 identity matrix is used as covariance matrix also in this case.

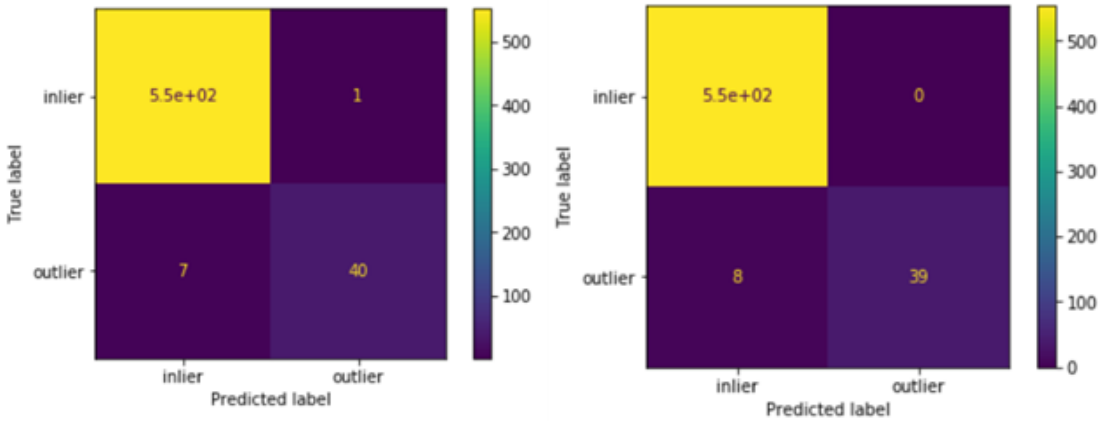
A total of 3000 instances are then spit into train and test sets with percentages of 80% and 20%, respectively.

The train set is given in input to MetaOD, that provides as top model LODA with $n\_bins = 15$ and $n\_random\_cuts = 150$. Also in this case, it is compared with the IF and the contamination is set to 0.1 for both models.

The metrics adopted are precision, recall, and F1 score and are visible in Table 2.1. Even if the performance are substantially equal in term of F1-score, LODA shows a better Precision, while a worst Recall, than IF. For completeness, Figure 2.20 shows the confusion matrices of IF (Figure 2.20a) and LODA (Figure 2.20b).

Model	**Precision**	**Recall**	**F1-score**
Isolation Forest	0.975	0.851	0.909
LODA	1.0	0.829	0.906

Table 2.1: IF and LODA performance comparison



(a) (b)

Figure 2.20: IF and LODA confusion matrices.

2.3.5 Conclusions about MetaOD

Basically, MetaOD exploits meta-learning to circumvent the problem of the model selection for an unsupervised OD dataset. By looking at the historical supervised datasets with which the meta-learner is trained and comparing them to a new dataset in terms of meta-features, it is able to find the most suitable model for the task at hand. However, the cleverness of this solution collides with the problem of the assumption on which MetaOD is based, i.e., that the similarity in terms of meta-features is enough to find a good model. The Anomaly detection task can be found in many different domains, each one characterised by a peculiar conception of what is abnormal and what is not. For this reason, OD datasets may differ greatly and even in the same domain. In particular, the 62 historical datasets that are used for training the meta-learner come from fields that are rather different from the industry, such as the medical one. For this reason, assessing the similarity between the new dataset and the historical ones may be a too weak criterion to find a suitable model. A possible solution consists in computing the distribution of the meta-features vectors distances of the historical datasets and compare it with the distances of the new dataset and the historical ones. If the new dataset results to be close enough (a threshold should be defined) to the historical ones (with respect to the distribution), then the MetaOD assumption can be considered reliable. Otherwise, the Isolation Forest can be used as a valid alternative.

2.4 Interpretability

2.4.1 Accelerated Model-agnostic Explanations

Providing interpretability for Machine Learning models is an important feature in particular for Anomaly Detection tasks. Indeed, it can be used as a root-cause analysis tool because of its ability to provide insight into which features have the greatest impact in determining a point to be classified as abnormal. Different anomalies may have different causes, so, from a granularity perspective, the local interpretability is preferred, since able to provide explanations that are instance-specific. Moreover, local intepretability can be exploited by companies to take the best corrective actions to avoid the occurrence of similar anomalies in the future.

Since MetaOD can provide as output a model among several different ones, it is not possible to consider an interpretability tool specifically designed for a given detector, but only a model-agnostic one.

Accelerated Model-agnostic explanations [10] is a model-agnostic interpretability approach that works both at the global and local level. It operates on tabular data and can be applied to both regression and classification models. In addition to providing feature importance scores, it also allows a *what-if* analysis useful for assessing how changes in features values would affect model predictions [10]. In addition, it is a computationally efficient method.

2.4.2 Interpretability with AcME

AcME is adopted to interpret the results of some abnormal instances of the CINECA Dataset. In Figure 2.21, a first example is shown. It is the local interpretability of an instance classified as abnormal by the Isolation Forest model. On the x axis there is the prediction, i.e., the anomaly score, that is 0.166 for the instance 941. As visible, a positive score identifies an outlier, while a negative score is assigned to a normal point. The y axis shows instead the features, ordered by global importance. This plot allows to assess how changes in features values impact on the predicted score. In particular, for each feature, AcME computes the quantiles of the empirical distribution of the feature. The coloured bar on the right of Figure 2.21 describes the different quantile levels, starting from low in blue and ending with high in red. Then, the *what-if* analysis is performed by changing the value of a feature, while keeping the others fixed, and seeing how the model prediction evolves. In the plot, the larger circles refer to the quantiles in which the features' values fall, while the smaller circles describe the other quantiles, allowing us to assess how the anomaly score changes. For instance, observation 941 has a very low value of percentage of jobs ended in the *COMPLETED* state (quantile 0.04, visible as the big blue circle on the second row), which is consistent with what one would expect from an anomalous instance. When increasing the value of any amount, we can see a reduction in the anomaly score. At the same time, by looking at the row relative to features *mean_queuetime*, it is possible to observe that the instance falls in the maximum quantile and by reducing the feature value, we reduce the abnormality of the observation. The high quantiles of features like the percentage of jobs *CANCELLED by USER*, *PREEMPTED*, *CANCELLED*, or

failed due to *NODE FAIL*, are as expected by an instance classified as abnormal.

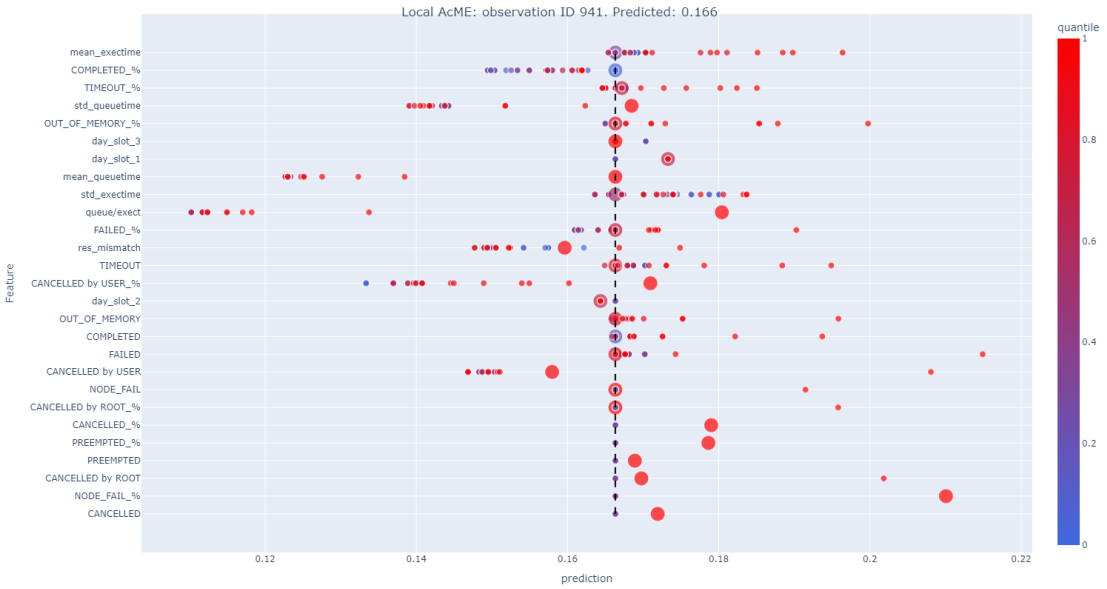


Figure 2.21: Local interpretability of sample 941 from CINECA Dataset. It is abnormal according to IF.

Another example is shown in Figure 2.22. In this case, it is shown an observation that is slightly abnormal (with a score of 0.03), according to the IF. Considerations similar to the previous example can be made, for instance about the low percentage of *COMPLETED* jobs and the high *mean_queuetime*. In this case, however, it is possible to see on which features to take action in order to make the observation normal. Indeed, given that zero is the threshold under which an instance is no longer considered abnormal, by reducing *mean_exectime* and/or the percentage of *TIMEOUT* jobs, that belong to very high quantiles (1 and 0.979, respectively), it is possible to change the instance's class.

The two examples of Figures 2.23 and 2.24 are relative to observations 2913 and 2963, respectively, which are classified as abnormals by LODA. The threshold is identified by the left vertical dashed line in both examples. In this case the interpretation is less clear, since it seems that the prediction is not altered for most quantiles of the majority of the features. However, for Figure 2.23, reducing the *mean_queuetime* (and the *std_queuetime*), makes the instance normal, similar to the example of observation 941 (Figure 2.21). Also reducing the *res_mismatch* has a similar effect. Instead, for Figure 2.24, a similar role is played by the

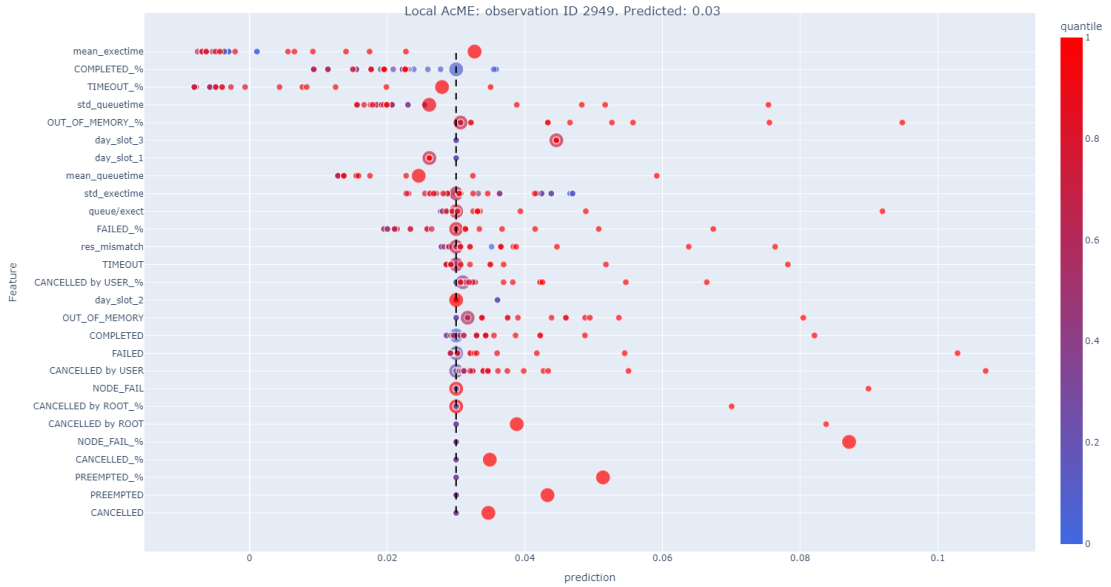


Figure 2.22: Local interpretability of sample 2949 from CINECA Dataset. It is slightly abnormal according to IF.

mean_exectime (and the *std_exectime*), while reducing the resources mismatch is not enough.

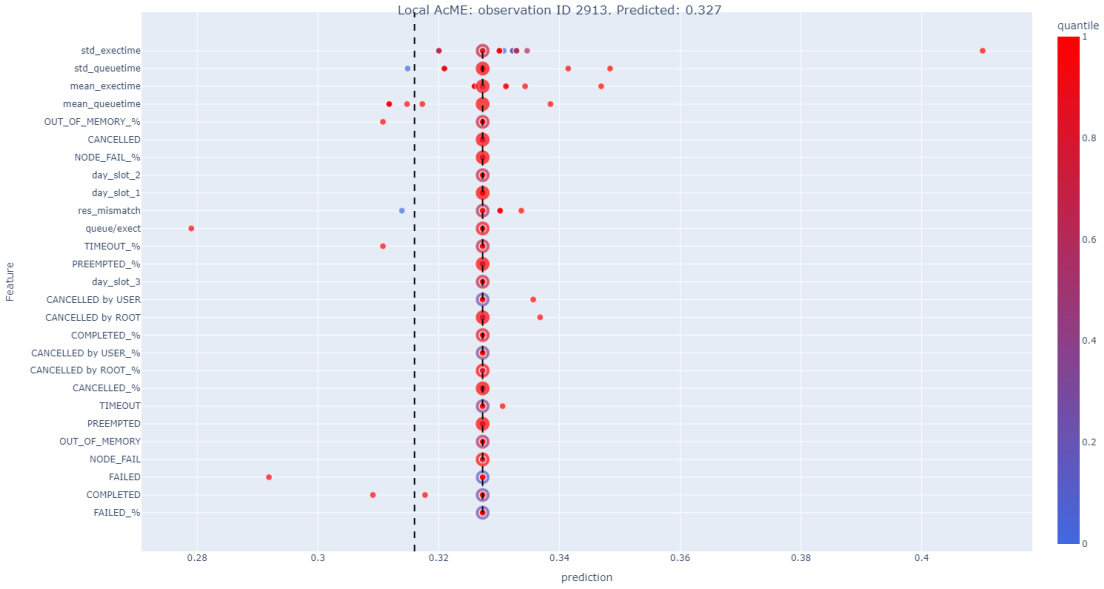


Figure 2.23: Local interpretability of sample 2913 from CINECA Dataset. It is slightly abnormal according to LODA.

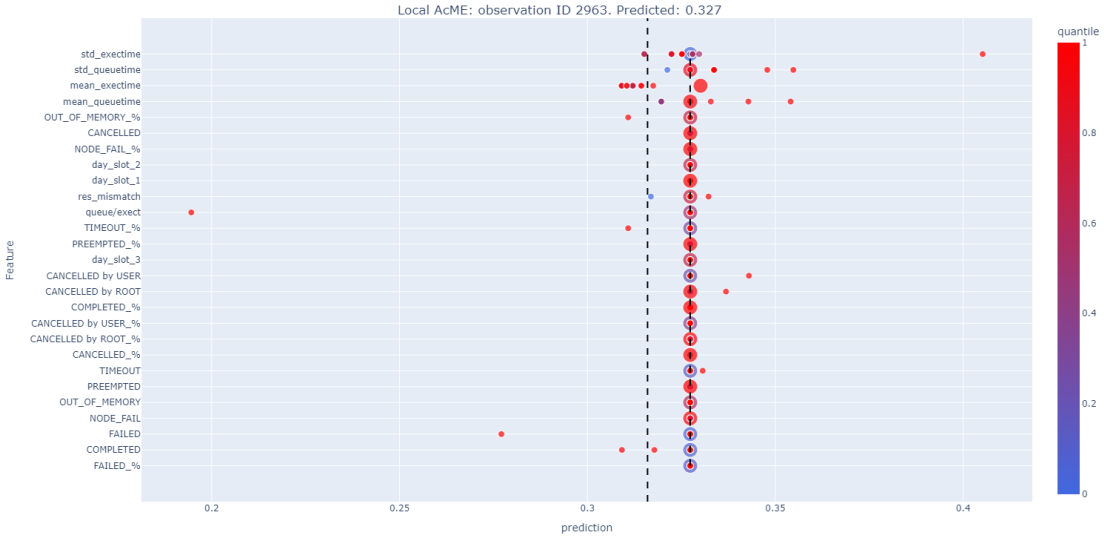


Figure 2.24: Local interpretability of sample 2963 from CINECA Dataset. It is slightly abnormal according to LODA.

3

Visual Anomaly Detection

3.1 Introduction

Visual anomaly detection is an important task involving machine learning and computer vision that consists in detecting abnormal patterns in images, i.e., those differing significantly from the normal ones. In many application scenarios, such as the industrial ones, this task is treated as an unsupervised one, mainly due to the lack of supervised data because abnormal images/patterns are usually variable in shape, color and size and they do not have stable statistical laws [61]. Moreover, annotating images, especially pixel-wise, is a time-consuming task, so it is rarely done in industry, especially in the early stages of the transition to digitisation.

As noted in [61], visual anomaly detection can be divided into image-level and pixel-level according to the visual detection granularity considered. They can be seen as binary classification tasks, where in image-level the focus is on classify images into normal or abnormal, while in pixel-level the classification regards the single pixels, that can belong or not to abnormal regions of the image (the result is an image segmentation). After the great success of deep convolutional networks, the researchers started to focus on deep learning techniques to deal with this challenging tasks. In particular, very different approaches are being considered, grouped in different categories according to common taxonomies, as in [61] and [55]. However, a broader categorisation is done in [62] and [12], where the authors distinguish between reconstruction-based and

representation-based methods. Reconstruction-based methods usually consider models such as autoencoders, generative adversarial networks, like GANomaly [3], or normalising flows, like Fastflow [62]. These models are used to encode and reconstruct the normal data, relying on the intuition that anomalies cannot be reconstructed since they are not included in the training data: they lead to a larger reconstruction error. Representation-based methods, such as Padim [12] or Patchcore [49], employ deep pre-trained convolutional neural networks to extract discriminative features from normal images/patches and then model the distribution of these features. The anomaly score for a test image is obtained calculating the distance between the test features and the modeled distribution. It is important to note that, during training, almost all these neural-based methods involve only normal, i.e., not anomalous, images or patches, so the visual anomaly detection task is usually formulated as a one-class learning problem for the unexpectedness of anomalies [60].

The following work on Visual Anomaly Detection focuses on identifying a suitable model for an AutoML library. Indeed, given the unsupervised nature of this task, there is no way to perform model selection and hyperparameters tuning for a given dataset. Up to now, the only way to implement an AutoML Visual AD module consists of selecting a model sufficiently flexible and strong to work well with most datasets. For this reason, several state-of-the-art neural network-based models are compared according to certain criteria in Section 3.4. These criteria were defined taking into account the needs of an AutoML library, for instance, in terms of performance, results' interpretability, and training speed.

The datasets considered for the experiments are described in Section 3.2, while the models and the adopted library are described in Section 3.3.

3.2 Datasets

3.2.1 Wood Anomaly Detection Dataset

In the Wood Anomaly Detection dataset [43], images consist of several views from wooden textured objects from the same class. There are four possible types of anomalies for the objects: crack, stain, porosity, and knot. The anomalous regions were manually labelled at pixel level by two different teams. The dataset is provided by the EUHubs4Data catalogue and consists of 619 normal train images, 183 normal validation images, 151 normal test images, and 327 abnormal

test images (the latter supplied with their respective binary masks). Therefore, the anomaly detection task is seen as a one-class classification task, where the model learns exclusively from non-abnormal wood textures. Some examples are visible in Figure 3.1.

(a) (b) (c)

Figure 3.1: Example of images from the Wood AD dataset.

3.2.2 MVTec Anomaly Detection Dataset

The MVTec Anomaly Detection dataset [6] is a widely used dataset to benchmark anomaly detection methods in an industrial scenario. It includes several categories of objects and textures. The ones considered in the experiments are "bottle", "screw" and "toothbrush", organised as follows:

- "bottle": 209 normal train images, 20 normal test images, and 63 abnormal test images. Examples are given in Figure 3.2.
- "screw": 320 normal train images, 41 normal test images, and 119 abnormal test images. Examples are given in Figure 3.3.
- "toothbrush": 60 normal train images, 12 normal test images, and 30 abnormal test images. Examples are given in Figure 3.4.

The abnormal images are organised into different typologies and are all provided with the corresponding binary masks.

3.3 State-of-the-art

Experiments carried out involve all state-of-the-art models implemented in the Anomalib [4] library: CFlow [25], DFM [2], DFKDE, Fastflow [62], Patchcore [49], Padim [12], STFPM [60] and GANomaly [3]. Anomalib is a deep learning library that includes implementations of state-of-the-art Visual AD algorithms and provides tools for benchmarking them on both public and private datasets.

(a) (b) (c)

Figure 3.2: Example of images from the "bottle" category of the MVTec AD dataset.

(a) (b) (c)

Figure 3.3: Example of images from the "screw" category of the MVTec AD dataset.

(a) (b) (c)

Figure 3.4: Example of images from the "toothbrush" category of the MVTec AD dataset.

Models DFM, DFKDE, and GANomaly perform only image-level anomaly detection, providing for each test image an anomaly score, while the others are more fine-grained, being able to segment the anomaly region. These last ones provide as output an image anomaly score, a heatmap highlighting the anomaly region, and a binary mask.

All models, excepting GANomaly, require defining a backbone, i.e., a convolutional neural network, usually pre-trained on ImageNet dataset. This neural network is used for extracting discriminative features from the normal images during training and from normal and abnormal images during testing. Very often, these backbones are of the Resnet family, like Resnet-18 or wide Resnet-50, or even EfficientNet.

In the following subsections, the models are briefly described.

3.3.1 Student-Teacher Feature Pyramid Matching

The Student-Teacher Feature Pyramid Matching (STFPM) [60] is a method that adopts the Student-Teacher learning framework to deal with the Visual AD task. As visible in Figure 3.5, there are two neural networks, called Teacher and Student, that have exactly the same architecture, usually a ResNet-18. The Teacher is pre-trained on ImageNet and its weights are frozen, while the Student is trained to imitate the behaviour of the Teacher. Since deep neural networks generate a pyramid of features from the input image [60], the idea of this method is to perform a training with the goal to match the features extracted by a few successive bottom layers groups of the Student with the corresponding ones of the Teacher [60]. These layers group are usually the residual blocks of a ResNet. The l_2 distance is adopted to measure the mismatch between the features of the two NN.

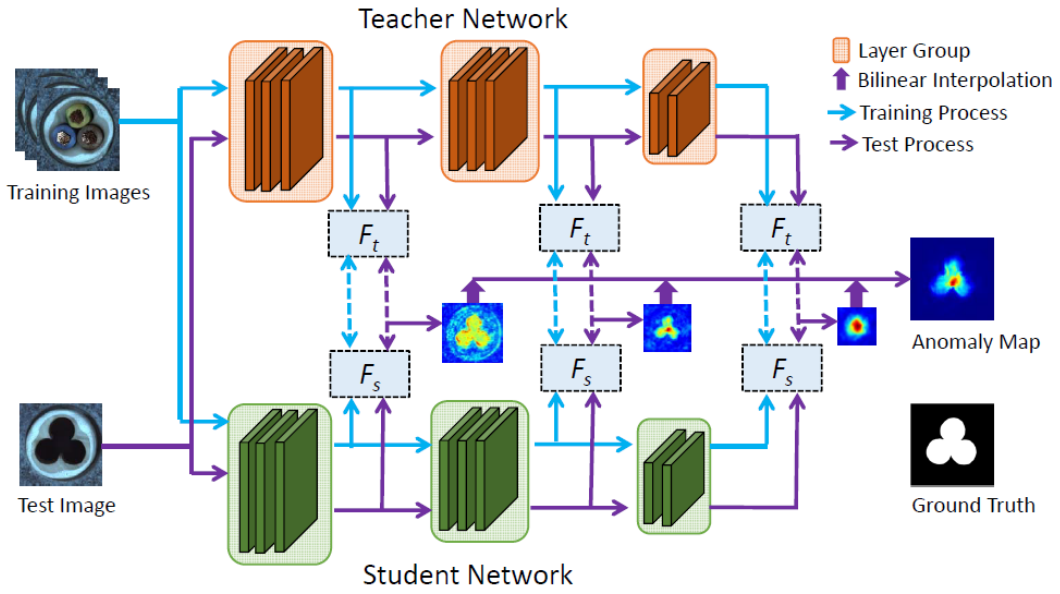


Figure 3.5: Schematic overview of the STFPM method. The feature pyramid of a student network is trained to match with the counterpart of a pre-trained teacher network. A test image (or pixel) has a high anomaly score if its features from the two models differ significantly. The feature pyramid matching enables the method to detect anomalies of various sizes with a single forward pass [60].

Since during training only normal images are involved, the capability of the Student to emulate the Teacher is good only in the cases without anomalies.

However, in the testing phase, the presence of also abnormal images leads to larger errors between the layers groups feature maps of the two models, since the Student deals with never seen instances. This error is exploited to compute an anomaly map. Then, an image is classified as abnormal if any pixel in the image is anomalous.

3.3.2 Cflow and Fastflow

Cflow [25] and Fastflow [62] both exploit the so-called normalising flows to estimate the distribution of the normal features, i.e., the ones extracted from the normal images. As defined in [47], a normalising flow (NF) describes the transformation of a probability density through a sequence of invertible mappings. In this case, it is a neural network used to learn how to transform the raw feature distribution of the normal images into a simpler distribution, i.e., the normal one. However, this transformation has an important property since the process is bijective. So the NF can be traversed in both directions. In Figure 3.6, it's visible the pipeline of the Fastflow method, with an highlight of the layers that compose the model. Notice that the features are extracted with a pre-trained CNN, like a ResNet. This methodology allows one to compute a heatmap by computing the likelihoods of the test image features according to the normal distribution: abnormal images should have lower probabilities.

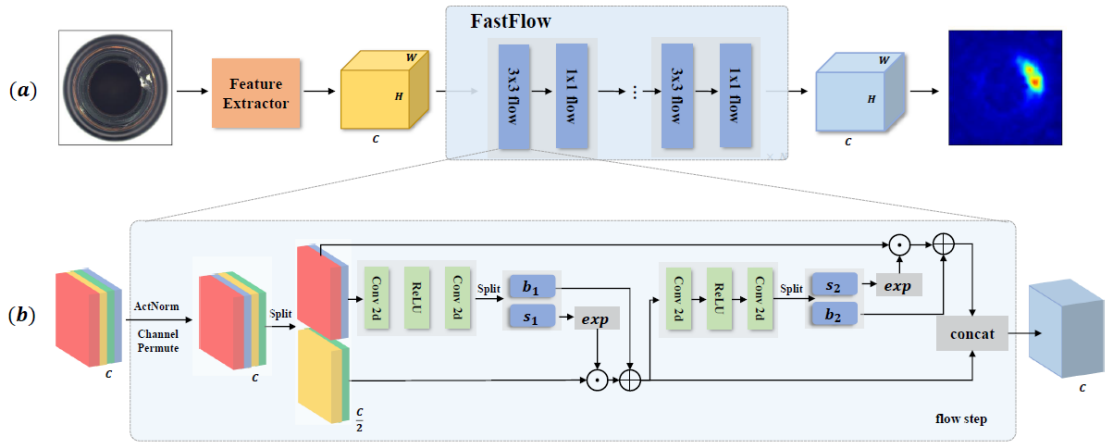


Figure 3.6: (a) the whole pipeline for unsupervised anomaly detection and localization of the method, which consists of a feature extractor and the FastFlow model. An arbitrary network can be used as the feature extractor such as CNN or vision transformer. FastFlow is alternatly stacked by the "3×3" and "1×1" flow. (b) one flow step for FastFlow, the "Conv 2d" can be 3×3 or 1×1 convolution layer for 3×3 or 1×1 flow, respectively. [62].

Cflow is a similar out-of-distribution (OOD) detector but with the important difference that as inputs for the flow there are not only the features, but also conditional inputs that encode information about the positions of the features. These new inputs are the 2D form of the conventional positional encodings (PE) [47]. The overview of this method is in Figure 3.7.

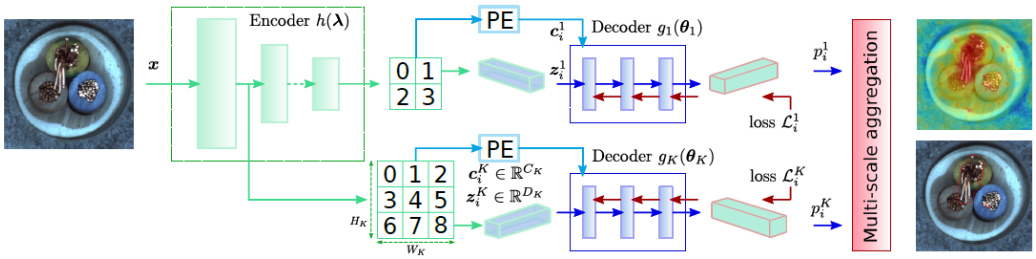


Figure 3.7: Overview of CFLOW-AD with a fully-convolutional translation-equivariant architecture. Encoder $h(\lambda)$ is a CNN feature extractor with multi-scale pyramid pooling. Pyramid pooling captures both global and local semantic information with the growing from top to bottom receptive fields. Pooled feature vectors z_i^k are processed by a set of decoders $g_k(\theta_k)$ independently for each kth scale. The decoder is a conditional normalizing flow network with a feature input z_i^k and a conditional input c_i^k with spatial information from a positional encoder (PE). The estimated multi-scale likelihoods p_i^k are upsampled to the input size and added up to produce anomaly map. [25].

3.3.3 DFM, DFKDE and GANomaly

The three models described in this Subsection perform an image-level classification, so they are not able to provide an anomaly map.

Deep Features Modeling (DFM) [2] and Deep Features Kernel Density Estimation (DFKDE) works similarly. They are characterized by a feature extraction stage, where, as usual, a deep pre-trained neural network is adopted as backbone and receives as input only normal images. Then, in the classification stage, the distribution of the normal features is learnt. The anomaly scores for the test instances are given by the negative log-likelihood under the learnt density models. For DFM, the second stage consists of a PCA for dimensionality reduction followed by a Gaussian Density Estimation, while in DFKDE a Gaussian Kernel Density Estimation is adopted.

The approach of GANomaly is rather different and combines Generative Adversarial Networks (GAN) [24] with Auto-Enconders (AEs). The pipeline is visible in Figure 3.8. It is possible to identify three sub-networks in the model. The generator G consists of an autoencoder that receives as input the image x and learns to reconstruct it: the image is compressed into a vector z, which is then expanded in $\hat{x}$. A second sub-network, called E, is composed of a single encoder, with the same architecture as the endoder of G. It learns to compress the reconstructed image $\hat{x}$ into a vector $\hat{z}$ by minimizing the l_2 distance from the vector z. Finally, the last sub-network is the discriminator network D, that is trained to classify the input x as real and the output $\hat{x}$ as false [3].

The training consists in minimizing a weighted sum of the three losses shown in Figure 3.8, and involves only normal images. The anomaly score for a test image x is given by the l_2 distance between z and $\hat{z}$.

3.3.4 Patch Distribution Modeling Framework

Patch Distribution Modeling Framework (PaDiM) [12] is a representation-based method that adopts a pre-trained (on ImageNet) CNN to extract features from the images patches and then learns a matrix of Gaussian parameters for represent them distribution.

As visible in Figure 3.9, the normal images are divided into patches and fed into the CNN. In particular, only the first three layers are considered when adopting ResNet as backbone: indeed, these layers have enough high resolution

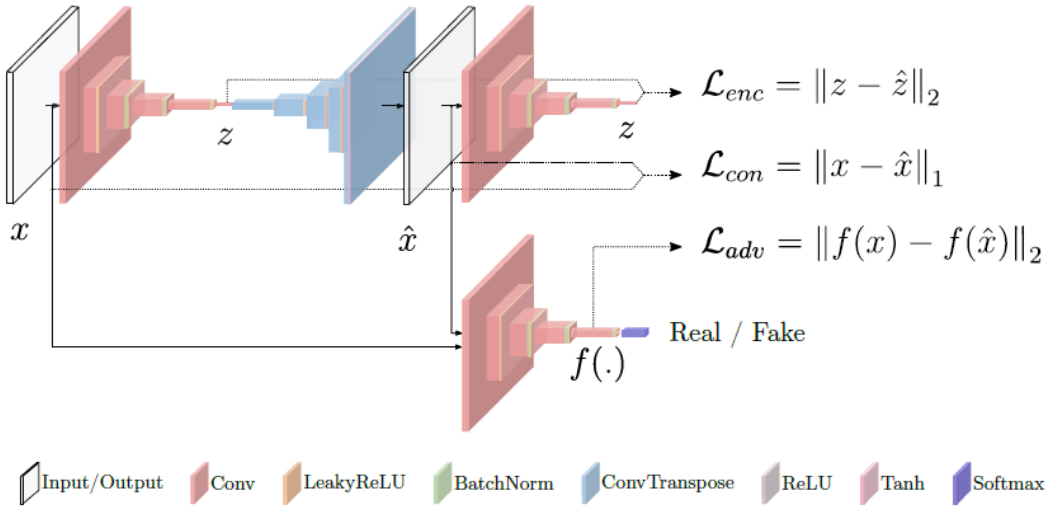


Figure 3.8: Pipeline of GANomaly [3].

and are not too biased towards ImageNet dataset. During training, for each patch of each normal image, three embedding vectors are computed, located in the activation maps of the three involved layers in the places corresponding to the patches' locations. These three vectors are concatenated and the resulting one is subjected to random subsampling to reduce the dimensionality. Padim works on the assumption that the features of each patch can be modeled by a multivariate Gaussian distribution. In fact, for each patch location (i, j) it learns the Gaussian parameters (μ_{ij}, Σ_{ij}) from the training embedding vectors of the corresponding patches. Then, the parameters of all these Gaussian distributions are used to fill a parameter matrix.

When a test image is provided to Padim, for each test patch, the model compute the Mahalanobis distance between its features and the Gaussian parameters of the corresponding location. The value obtained is an anomaly score. The score for an image is defined as the maximum score for the anomaly map.

3.3.5 Patchcore

Patchcore [49] is a representation-based method with some similarities to Padim. Indeed, also for this model, a pre-trained CNN (typically a ResNet) is used as feature extractor for the image patches, as shown in Figure 3.10. In the training phase, for each patch of each normal image, the corresponding features of two consecutive intermediate layers are considered and aggregated to obtain

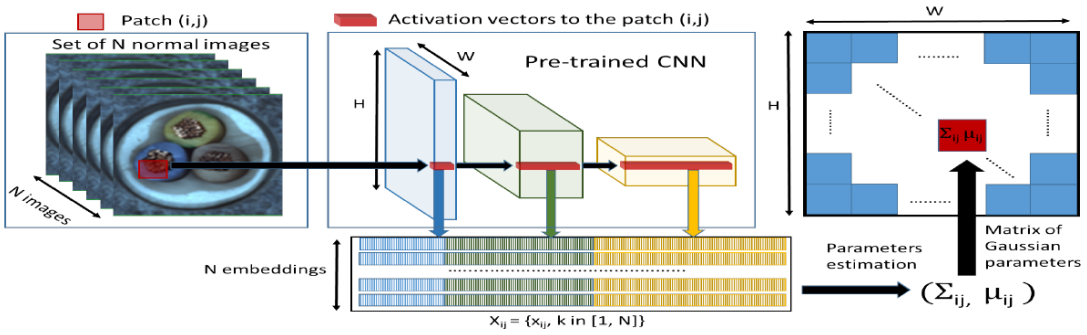


Figure 3.9: For each image patch corresponding to position (i, j) in the largest CNN feature map, PaDiM learns the Gaussian parameters (μ_{ij}, Σ_{ij}) from the set of N training embedding vectors $X_{ij} = \{x_{ij}^k, k \in [[1, N]]\}$, computed from N different training images and three different pretrained CNN layers [12].

a single vector. In order to increase the receptive field size, the feature vectors of the neighbors patches are aggregated to the vector of the considered patch, obtaining locally-aware patch features.

The vectors extracted during training populate a memory bank $\mathcal{M}$, that, however, results extremely large. For this reason, a minmax facility locations coreset subsampling is performed to reduce the memory bank size.

In the test phase, the images are divided into patches, and the corresponding neighbourhood-aware features are computed. The anomaly score of a single patch is given by the maximum distance between its feature and the element in $\mathcal{M}$, while the anomaly score for the whole image is the maximum of the single patches scores.

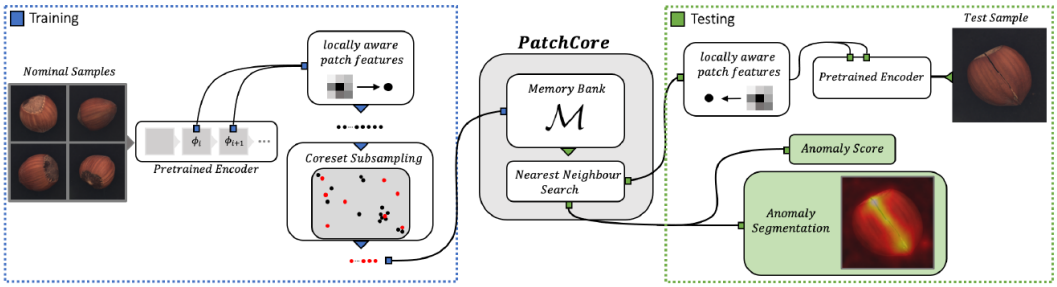


Figure 3.10: Overview of PatchCore. Nominal samples are broken down into a memory bank of neighbourhood-aware patch-level features. For reduced redundancy and inference time, this memory bank is downsampled via greedy coreset subsampling. At test time, images are classified as anomalies if at least one patch is anomalous, and pixel-level anomaly segmentation is generated by scoring each patch-feature [49].

3.4 Experiments

3.4.1 Comparison on Wood Dataset

The first models' comparison is done on the Wood dataset, with a Resnet-18 as backbone. All Wood images, resized to 256x256 pixels, are considered, excepting for Patchcore, where for computational reason the number of images is halved. The models' hyper-parameters are kept as defined in the Anomalib configuration files.

Four image-level metrics are used, that are: Precision, recall, F1 score, and the area under the Precision-Recall curve (AUPR); the latter is considered in place of AUROC as it is better suited for unbalanced classification tasks [11], [50]. The only pixel-level metric adopted is the AUPR, with a similar reasoning as for the image-level one: the anomaly region is usually minority compared to the rest of the image, so the pixel-level classification is strongly unbalanced. The results are shown in Table 3.1, which also includes the sizes of the neural networks and the weights of the models in MB.

Model	**Image Precision**	**Image Recall**	**Image F1-score**	**Image AUPR**	**Pixel AUPR**	**N. parameters**
Fastflow	0.701	0.991	0.821	0.696	0.187	9.7M(38MB)
DFM	**0.733**	0.964	**0.832**	**0.795**	-	2.8M(11MB)
DFKDE	0.686	**1.0**	0.814	0.771	-	11.2M(44.7MB)
Padim	0.689	0.99	0.813	0.65	0.143	2.8M(11MB)
STFPM	0.715	0.972	0.824	0.735	**0.322**	5.6M(22MB)
GANomaly	0.703	0.982	0.82	0.771	-	188M(755MB)
Cflow	0.684	**1.0**	0.812	0.614	0.155	25.3M(101MB)
Patchcore	0.707	0.994	0.827	0.771	0.274	2.8M(11MB)

Table 3.1: Anomalib models comparison on Wood dataset (in bold the best results for each metric) .

The best performance in terms of Image AUPR is of DFM, that is, however, only able to perform image classification, and does not provide any heatmap. For an AutoML library, the capability to provide to the user a detailed heatmap highlighting the anomalous region of an image is a very important feature. This focus on the interpretability of the results leads to exclude this model from further comparisons, and similarly, DFKDE and GANomaly are no longer

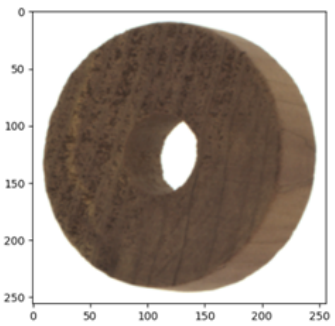
considered. GANomaly has also another strong drawback, that is, the huge Neural Network size (188M parameters).

The quality of the heatmap is another important way to compare visual anomaly detection models. In Figures 3.11a and 3.12a there are comparisons of the heatmaps from Fastflow, Padim, STFPM, CFlow and Patchcore for two anomalous test images of Wood dataset. In Figure 3.11, the anomalous region is wide, covering almost half of the object, as confirmed by the mask in Figure 3.11b. Instead, the wood defect of Figure 3.12a is concentrated in a small area (see mask in Figure 3.12b).

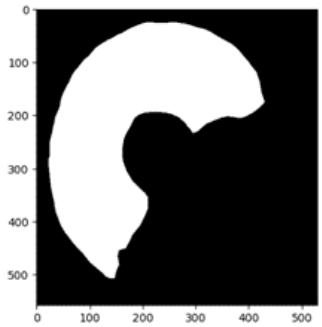
The most promising results are of STFPM, Padim and Patchcore, that seem to localize quite well the anomalous region. Fastflow can detect the defect of Figure 3.12a, but the heatmap provided in Figure 3.11e is almost completely wrong. At the same time, CFlow is not very accurate and, moreover, the output is not so interpretable due to the low contrast between the anomalous region and the normal one. As can be seen in Figure 3.12d, STFPM return the most precise results and, in general, more readable heatmaps. This is consistent with the best result of 0.322 in the Pixel AUPR metric.

Another important feature that is required in an AutoML library is to have fast training and testing phase. For this reason, CFlow is excluded due to the relatively slow training time (about 1 hour). Among the remaining models, it is important to observe that Padim and Patchcore do not require a proper training phase. Indeed, they leverage on the pre-trained backbones to extract features from the normal images. Then, in Padim, features are used to model patch-wise multivariate Gaussian distributions, useful for assess if the patches of a test image are anomalous or not; in Patchcore, instead, normal features populate a memory bank, that is used to score each test patch. This implies that there is no proper update of the parameters, and the two models require only a single epoch to train. However, a bottleneck for Patchcore is represented by the coreset subsampling of the feature set, done before populating the memory bank.

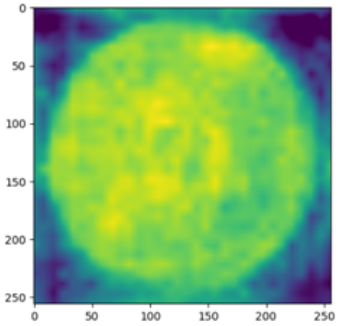
Differently from these two models, Fastflow and STFPM require a more classical training phase. Fastflow adjusts the parameter of the 2D normalising flow used as probability distribution estimator, while in STFPM the Student must learn to reproduce the output of different residual blocks of the pre-trained Teacher.



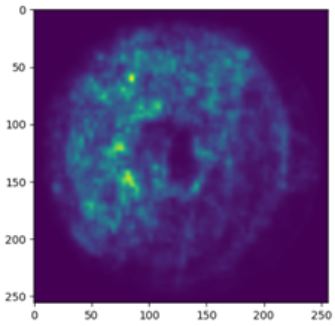
(a)



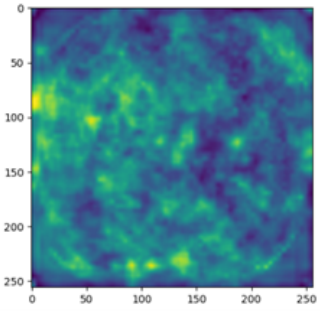
(b)



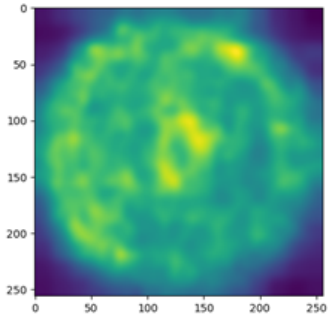
(c)



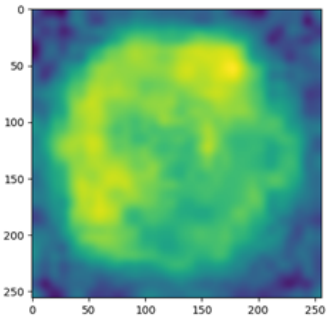
(d)



(e)

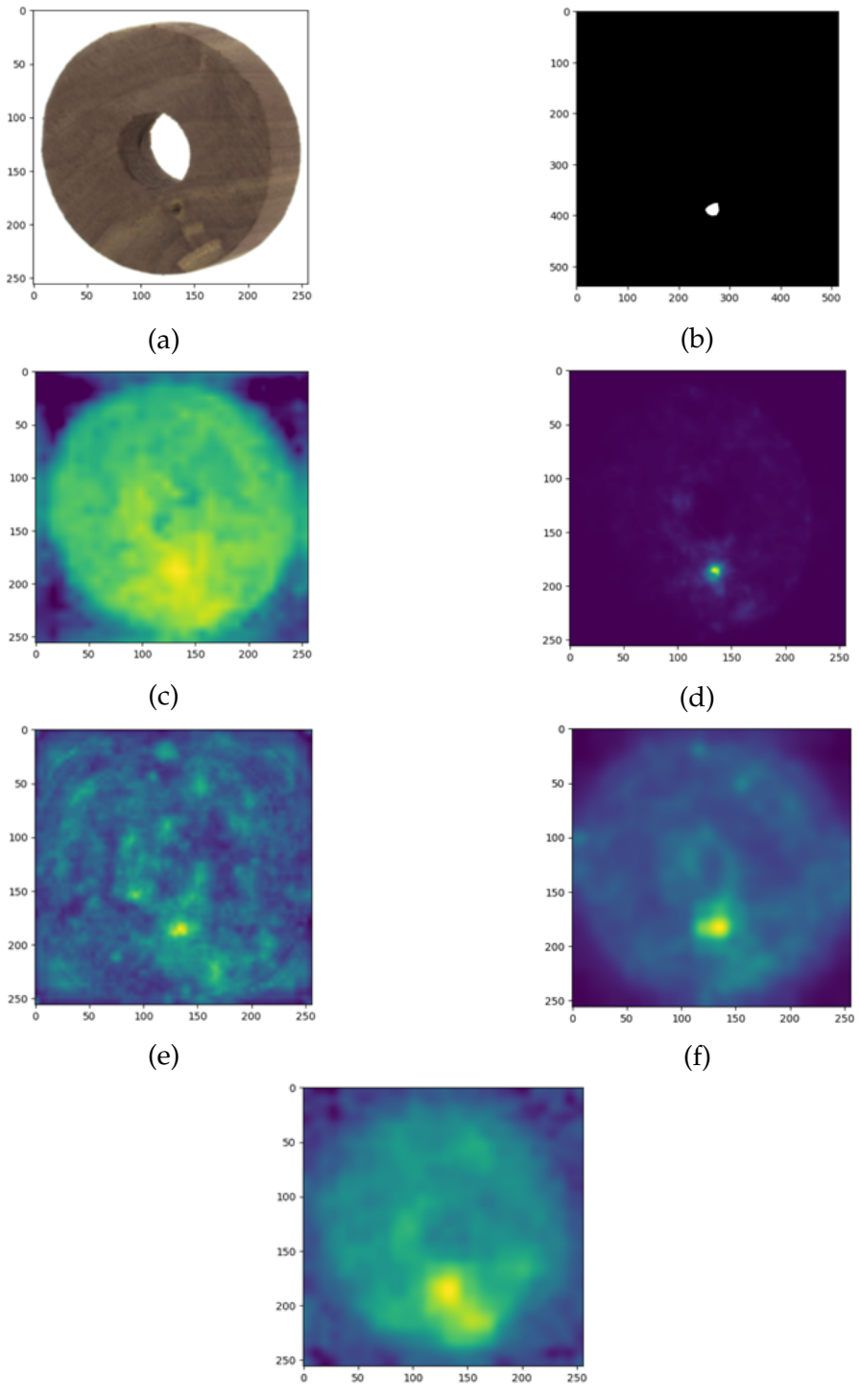


(f)



(g)

Figure 3.11: In 3.11a an example of image with wide anomalous region from the Wood dataset and in 3.11b the corresponding mask. Heatmaps of Cflow, STFPM, Fastflow, Padim, and Patchcore are shown in Figures 3.11c, 3.11d, 3.11e, 3.11f and 3.11g, respectively.



(a) (b) (c) (d) (e) (f) (g)

Figure 3.12: In 3.12a an example of image with narrow anomalous region from the Wood dataset and in 3.12b the corresponding mask. The heatmaps of Cflow, STFPM, Fastflow, Padim and Patchcore are in Figures 3.12c, 3.12d, 3.12e, 3.12f and 3.12g, respectively.

3.4.2 Padim vs Patchcore

Taking into account both heatmap results and time requirements for training and testing, the selected models for further comparisons are Patchcore and Padim. In Table 3.2 are visible the results of Patchcore and Padim on the Wood dataset, considering only half of the data, and with two different backbones, Resnet-18 and Wide Resnet-50. Notice that now also Precision and Recall at the pixel level are included, for a more detailed analysis.

Model	**Image Precision**	**Image Recall**	**Image F1-score**	**Image AUPR**	**Pixel Precision**	**Pixel Recall**	**Pixel AUPR**
Padim R-18	0.684	**1.0**	0.813	0.661	0.192	0.624	0.18
Padim WR-50	0.688	**1.0**	0.815	0.662	0.168	**0.701**	0.157
Patchcore R-18	**0.696**	0.982	0.814	0.762	0.261	0.568	0.25
Patchcore WR-50	0.692	0.994	**0.816**	**0.783**	**0.276**	0.543	**0.275**

Table 3.2: Padim vs Patchcore on Wood dataset with two different backbones: Resnet-18 (R-18) and Wide Resnet-50 (WR-50) . In bold the best results for each metric.

According to the overall metrics Image AUPR and Pixel AUPR, Patchcore outperforms Padim with both Resnet-18 and wide Resnet-50. However, Padim shows a better Recall at pixel and image levels. Moreover, the training and testing times of this latter model are much better (see Table 3.3), since Padim is not slowed down by a coreset subsampling, like in Patchcore, but performs only a random feature selection. The train and test times of Table 3.3 are obtained with half of the Wood dataset. Note that with wide Resnet-50 both models have 24.9 M parameters (about 100 MB).

Model	**Training time**	**Testing time**
Padim R-18	20s	1m
Padim WR-50	1m 10s	1m
Patchcore R-18	7m	1m 15s
Patchcore WR-50	7m 20s	1m 25s

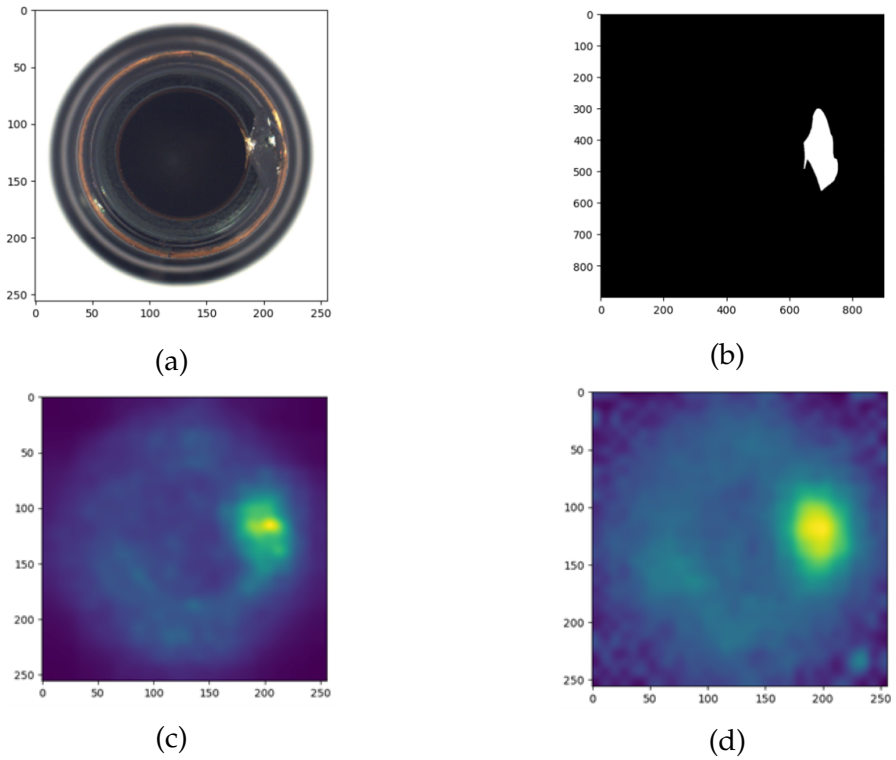
Table 3.3: Training and testing times comparison between Padim and Patchcore on half Wood dataset.

In Table 3.4, the two models are compared on the "bottle" category of the MVTec dataset, both showing remarkable results in the image-level metrics. The pixel-level ones are much better with respect to previous dataset, but still not

outstanding, proving that obtaining a precise anomaly segmentation is a difficult task even with state-of-the-art models and a relatively easy dataset. Figures 3.13a and 3.14a show two images of the MVTec dataset with the corresponding masks, together with the heatmaps generated by Padim and Patchcore, when considering Resnet-18 as the backbone.

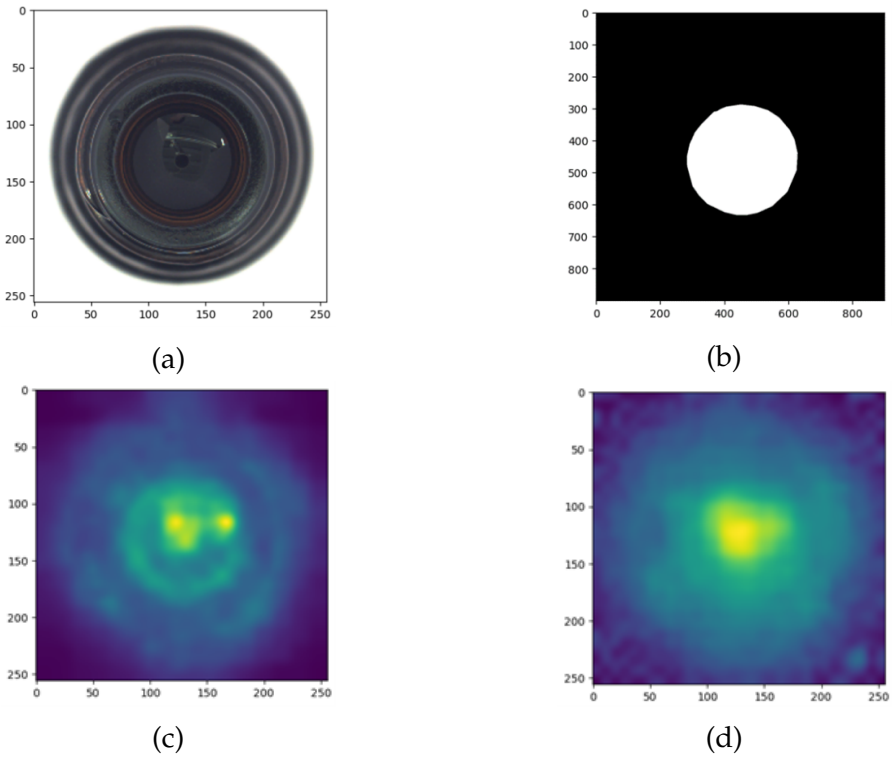
Model	**Image Precision**	**Image Recall**	**Image F1-score**	**Image AUPR**	**Pixel Precision**	**Pixel Recall**	**Pixel AUPR**
Padim R-18	0.984	**1.0**	0.992	0.999	0.651	0.804	0.709
Padim WR-50	0.984	**1.0**	0.992	0.999	0.616	**0.827**	0.711
Patchcore R-18	**1.0**	**1.0**	**1.0**	**1.0**	0.647	0.759	0.722
Patchcore WR-50	**1.0**	**1.0**	**1.0**	**1.0**	**0.652**	0.761	**0.757**

Table 3.4: Padim vs Patchcore on the "bootle" category of the MVTec dataset with two different backbones: Resnet-18 (R-18) and Wide Resnet-50 (WR-50) . In bold the best results for each metric.



(a) (b) (c) (d)

Figure 3.13: In 3.13a an example of image with a small break from the "bottle" category of the MVTec dataset and in 3.13b the corresponding mask. The heatmaps of Padim and Patchcore with R-18 are shown in Figures 3.13c and 3.13d, respectively.



(a) (b) (c) (d)

Figure 3.14: In 3.14a an example of image with a contamination from the "bottle" category of the MVTec dataset and in 3.14b the corresponding mask. Heatmaps of Padim and Patchcore with R-18 are in Figures 3.14c and 3.14d, respectively.

3.4.3 Robustness to data numerosity

Given the best compromise between results' quality and computational efficiency, Padim turns out to be the model that is most suitable for an AutoML library. Other experiments, involving only this model, are designed to test how many train images are needed to maintain good performance. In Tables 3.4 and 3.2 there are the results of Padim on MVtec and Wood datasets, respectively, when considering only a portion of the original training set (e.g., 3/4, ½, ¼, ... of the original data). The considered backbone is Resnet-18.

Padim seems to work well even with large reductions of the training set (in particular, 1/20 in Table 3.4 and 1/80 in Table 3.2 include only 10 training images for MVTec and Wood datasets, respectively), showing great robustness to data numerosity. In Table 3.4, the most noticeable performance drops are in Image Recall and Image F1-score when considering at most 1/10 of the training data and in Image Precision with 1/20 of the training data. On Wood dataset, instead, the metrics more affected by data reduction are the pixel-level ones, in

Training data fraction	Image Precision	Image Recall	Image F1-score	Image AUPR	Pixel Precision	Pixel Recall	Pixel AUPR
3/4	0.984	1.0	0.992	0.999	0.658	0.793	0.702
1/2	0.984	1.0	0.992	0.998	0.659	0.802	0.729
1/4	0.984	1.0	0.992	0.999	0.648	0.813	0.722
1/10	0.983	0.937	0.959	0.989	0.659	0.75	0.706
1/20 (10 images)	0.954	0.984	0.969	0.996	0.66	0.726	0.711

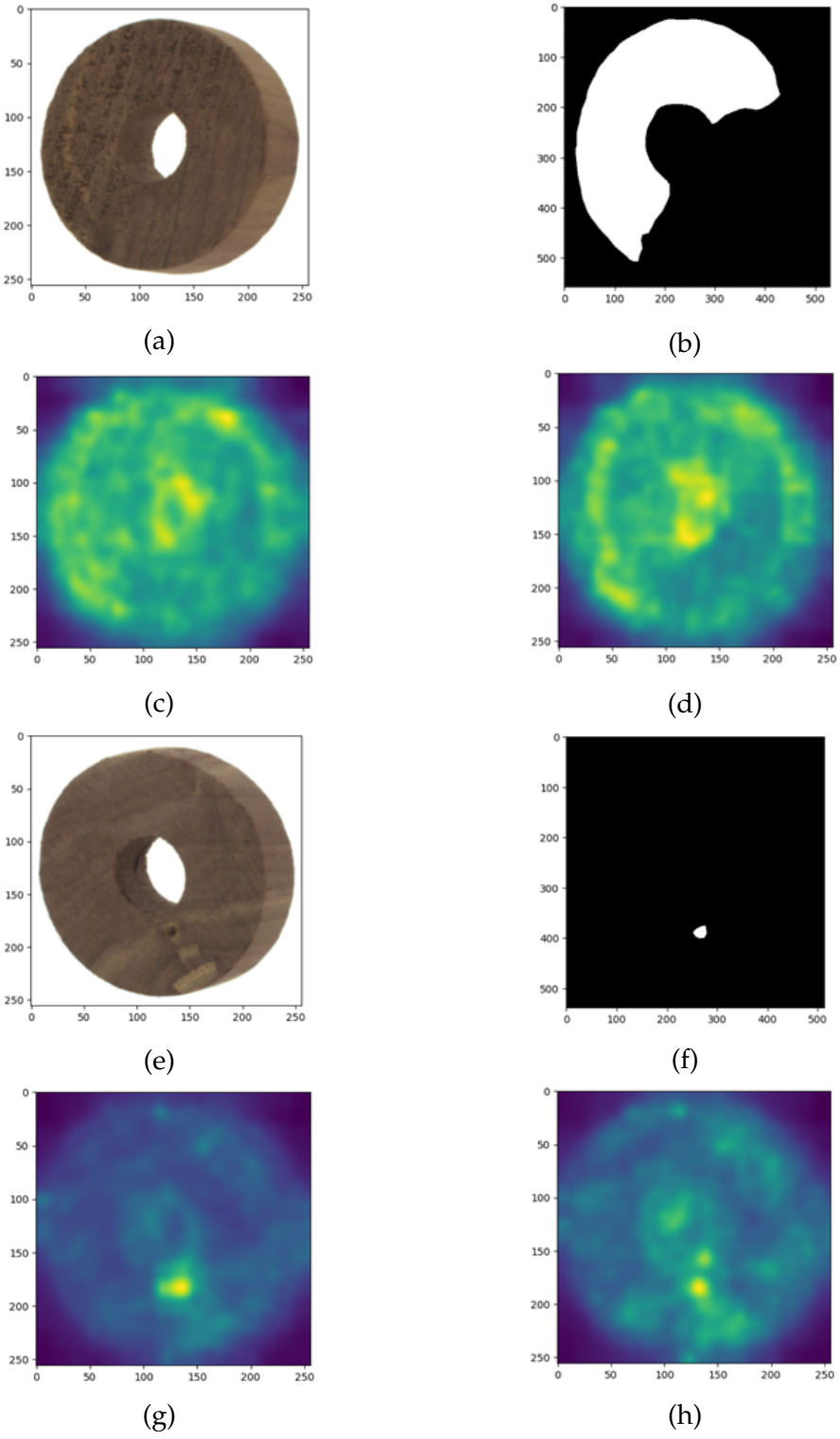
Table 3.5: Padim (R-18) performance on the category 'bottle' of the MVTec dataset with different traning set fractions.

Training data fraction	Image Precision	Image Recall	Image F1-score	Image AUPR	Pixel Precision	Pixel Recall	Pixel AUPR
3/4	0.685	1.0	0.813	0.647	0.159	0.747	0.153
1/2	0.688	0.996	0.814	0.66	0.171	0.663	0.167
1/4	0.684	1.0	0.812	0.672	0.155	0.72	0.147
1/10	0.684	1.0	0.812	0.637	0.156	0.728	0.145
1/20 (10 images)	0.687	0.994	0.812	0.645	0.13	0.85	0.112

Table 3.6: Padim (R-18) performance on the Wood dataset with different traning set fractions.

particular Pixel Precision and Pixel AUPR.

In Figures 3.15 and 3.16 it is possible to see some heatmap comparisons. The images involved are the same as for the previous experiments, two from the Wood dataset and two from the "bottle" category of the MVTec dataset. The heatmaps returned by Padim after a training with 1/10 of the training images (Figures 3.15d 3.15h 3.16d 3.16h), are generally less precise than the original ones (Figures 3.15c 3.15g 3.16c 3.16g), often highlighting as abnormal regions that are not. However, the localization of the anomalous regions is broadly correct in all images, confirming the ability of Padim to provide good results even with relatively few training instances.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 3.15: In 3.15a and 3.15e there are the two example from "Wood" dataset with corresponding masks in 3.15b and 3.15f. Figures 3.15c and 3.15g are the heatmaps obtained training Padim with the full training set, while for Figures 3.15d and 3.15h only 1/10 of the training images are used.

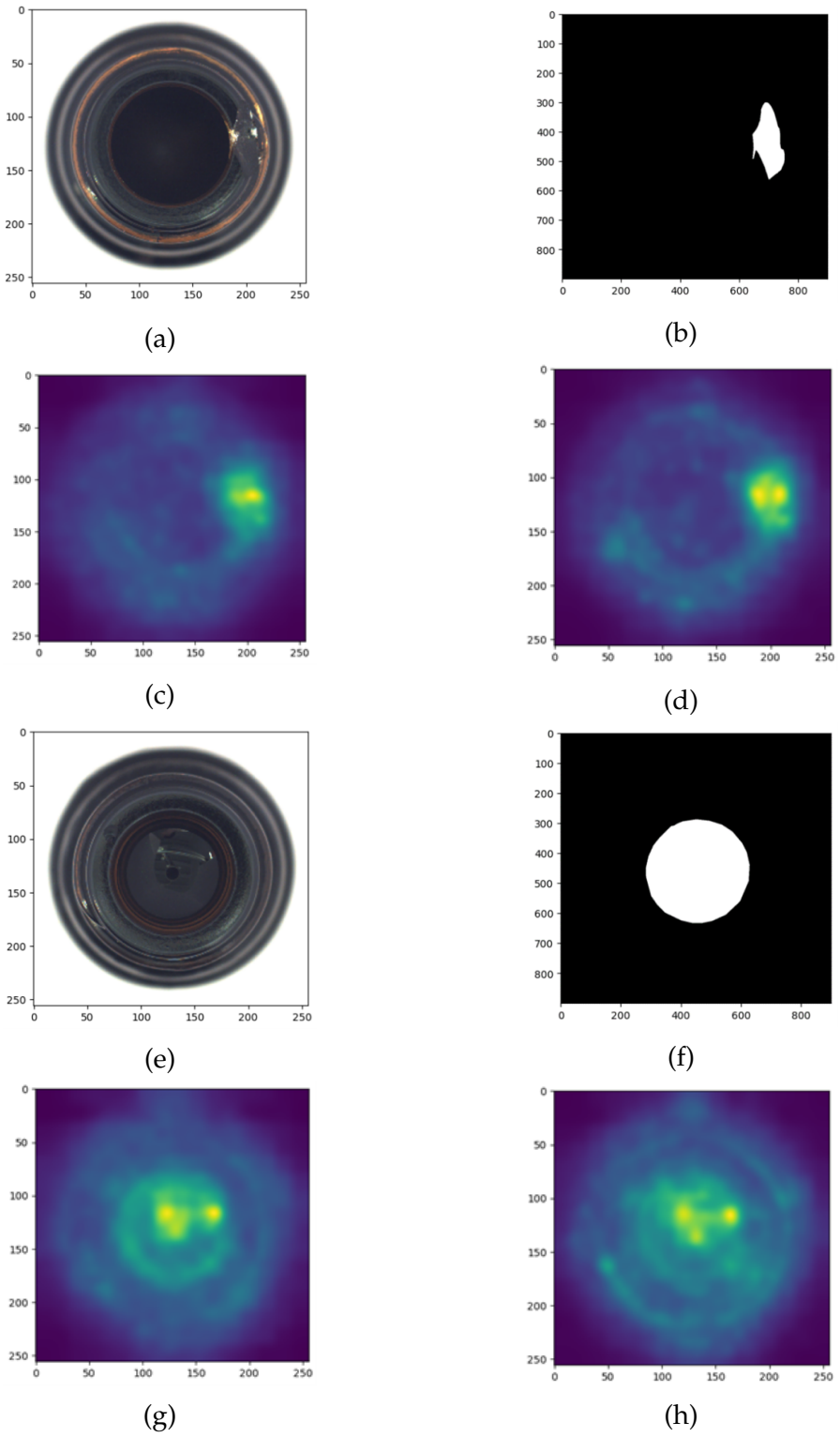


Figure 3.16: In 3.16a and 3.16e there are the two example from the 'bottle' category of the MVTec dataset with corresponding masks in 3.16b and 3.16f. Figures 3.16c and 3.16g are the heatmaps obtained training Padim with the full training set, while for Figures 3.16d and 3.16h only 1/10 of the training images is used.

3.4.4 Not aligned datasets

The last experiments are done to assess Padim behaviour with not aligned datasets, such as the "screw" folder of MVTec (see Figure 39). As observed in [55], the performance of this model degrades when images are not aligned, e.g., when the visualised objects are not at the same position or scale across different images. Indeed, during the training phases, Padim learns a multivariate Gaussian distribution for each different fixed patch location. Then, when a test patch is evaluated by the model, it is compared only with the distribution related to the corresponding location. The assumption that the patch features can be modelled by a Gaussian distribution may be too strong when dealing with unaligned or rotated data.

Table 3.7 confirms this theoretical consideration, since Padim does not perform very well if compared to Patchcore on the “screw” category of the MVTec dataset, both at image and pixel levels.

Model	**Image Precision**	**Image Recall**	**Image F1-score**	**Image AUPR**	**Pixel Precision**	**Pixel Recall**	**Pixel AUPR**
Padim (R-18)	0.818	0.941	0.875	0.87	0.165	0.309	0.137
Patchcore (R-18)	0.88	1.0	0.937	0.977	0.353	0.449	0.345

Table 3.7: Padim (R-18) vs Patchcore (R-18) on the "screw" category of the MVTec dataset.

To further verify this, a last experiment considers the "toothbrush" category. In this case, train and test images are aligned and metrics for Padim and Patchcore on this data are available in the first two rows of Table 3.8.

Model	**Image Precision**	**Image Recall**	**Image F1-score**	**Image AUPR**	**Pixel Precision**	**Pixel Recall**	**Pixel AUPR**
Padim (R-18)	0.882	1.0	0.938	0.936	0.458	0.8	0.49
Patchcore (R-18)	0.909	1.0	0.952	0.976	0.563	0.668	0.586
Padim (R-18) alt	0.75	1.0	0.857	0.866	0.323	0.556	0.314
Patchcore (R-18) alt	0.935	0.967	0.951	0.977	0.566	0.676	0.59

Table 3.8: Padim (R-18) vs Patchcore (R-18) on the "toothbrush" category of the MVTec dataset before and after alteration.

To test the capability of the models in managing unaligned objects, the train images are altered by means of random translations and rotations. The Albumentations [8] library is adopted to horizontally shift images by a percentage

of the width randomly sampled in the range [-0.2, 0.2], while small rotations are performed similarly with degrees in [-20, 20]. Vertical translations are not considered to prevent the anomalous region from falling out of the image. After training Padim and Patchcore on these altered images, the evaluation is performed with the original test set, leading to the results visible in the last two rows of Table 3.8. Patchcore is not too affected by the alteration, showing Image and Pixel AUPR values almost equal to before. The only substantial changes are on Image Precision and Recall, the first of which even improves. Differently, Padim's performance significantly degrade in almost all metrics, confirming that dealing with misaligned images challenges the model.

3.4.5 Conclusions

In this final subsection the main results of the previous experiments are summarised. After an initial comparison on Wood dataset, DFM, DFKDE and GANonamly are easily excluded since unable to provide an explanation for their results, in terms of heatmaps highlighting the anomalous regions. The remaining five models, that are STFPM, Cflow, Fastflow, Padim and Patchcore, are instead further compared in terms of heatmaps' quality: Cflow and Fastflow provide the worst results, while the best are the STFPM ones. However, the absence of a proper training phase in Padim and Patchcore leads the analysis to select these two models as the most suited. Indeed, their training is faster and, as observed, in particular, for Padim, seems not to require too much data. The comparison on the "bottle", "screw" and "toothbrush" categories of the MVTec dataset shows that Patchcore tends to perform better than Padim, but, unfortunately, it is slowed down by the coreset subsampling. The main drawback of Padim is its difficulty with not aligned dataset.

In conclusion, the presented results show that Patchcore is a more reliable model, robust against translation and rotation of the objects in the acquired images. However, Padim is recommended for its fast training in the contexts where the image acquisition system is consistent in terms of image location and rotation.

4

Predictive Maintenance

4.1 Introduction

4.1.1 Predictive Maintenance and Remaining Useful Life Estimation

With the birth of the so-called Fourth Industrial Revolution, or Industry 4.0, several new concepts have emerged, such as Predictive Maintenance (PdM). Maintenance 4.0 is a recent preventive maintenance approach that leverages the large amount of data extracted from industrial equipment to increase its operational life and minimise costs due to machine downtime. Indeed, a growing number of companies nowadays invest in sensing technologies to monitor the status of their equipment, paving the way for data-driven maintenance solutions.

In the literature, maintenance policies are broadly categorised into Run-To-Failure (R2F), Preventive Maintenance (PM), and Predictive Maintenance (PdM). R2F is the simplest and costliest policy, but exposes you to unexpected breakdowns (with the associated costs in terms of lost production), since it consists of intervening only after the failure occurs. PM is also called scheduled maintenance because it is performed periodically; however, it is not as efficient as may seem, as some interventions may not be necessary. Pdm aims to optimise maintenance, performing interventions only when needed by estimating equipment health status in a data-driven way [1] [31] [54].

ML-based PdM can be divided into supervised and unsupervised PdM as

usual for all Machine Learning tasks [54]:

- **Supervised PdM**: the failures are provided within the dataset.
- **Unsupervised PdM**: dataset includes logistic and/or process information, but not failures related data.

Remaining useful life (RUL) estimation is an important PdM task that consists of predicting the remaining amount of time in which an equipment is operative, that is, it works as expected. The time can be measured in cycles, days, or other quantities. There are three main approaches to RUL prediction [31] [13]:

- **Physics-based** methods attempt to create a physical model of the equipment, but are computationally expensive and may not generalise well.
- **Data-driven** methods try to find a relationship between the monitored sensor data and the RUL, often exploiting Machine Learning.
- **Hybrid** methods try to combine the two previous approaches.

Moreover, the RUL estimation task can be treated as a classification or regression task. In the classification case, the single instances of the involved time series can be classified as faulty or non-faulty. For example, instances up to x days before the fault may be considered faulty, where x is a time horizon defined according to the predefined logistic and economic needs of the company. Instead, for regression, the goal is to predict the remaining lifetime value as accurately as possible, for each time instance. In this work, the supervised regression task is considered, and sensor data are used to train a regressor to retrieve the correct RUL value.

4.1.2 Remaining Useful Life Computation

Data sets for RUL estimation are usually provided as tabular data, where rows refer to single time instances, so they can contain information about cycles, days, hours, or any time unit considered. The columns store instead logistic and/or process information for each time instance. Usually, sensor measures are indicated, but other values regarding, for instance, the operational settings, may also be present. One of the columns should contain the target value, that is, the remaining useful life of the equipment at each time instance. Often, this column is not provided by the manufacturer together with the raw dataset. For this reason, the first issue to be solved when developing a PdM module for an AutoML library consists in defining utilities able to infer the RUL column.

There are different scenarios that can be considered. In the simplest case, each row is provided with the specification of the time instance to which it refers. For instance, in the NASA dataset described in Section 4.2, each row contains information about the cycle. Therefore, a simple utility function can be defined that checks for the maximum cycle value for the time series and then computes the RUL column for each instance by subtracting the cycle value to the maximum. The Ceruleo library provides a function of this type and uses it to infer the RUL columns for the NASA dataset.

However, the raw data provided by a manufacturing company may not contain the "high-level" information of the time instance, but instead only a timestamp, with indications about data, hour, and so on, regarding the acquisition. Clearly, given the supervised nature of the RUL estimation task, the manufacturer must provide a list of break events for the industrial equipment, together with the date of the event. In this scenario, the RUL can be computed from the difference between the breakage timestamp/date and the acquisition timestamp of the corresponding equipment. Nevertheless, there are two issues to consider. Firstly, the acquisition timestamp format may differ from the one of the breakage, so a proper conversion is needed, for example, reducing both timestamps to the same unit (seconds, hours, etc.). Second, the time granularity for the RUL must be defined, depending on the needs of the task being considered, which may require a level of accuracy in estimation of days, hours, or even seconds. Therefore, the RUL column obtained depends both on the quality of data and annotations (level of detail of the timestamps) and on the requirement of the specific domain in which the RUL estimation task is applied.

4.1.3 Problem Formalisation

A common approach to RUL estimation is to consider it as a classical regression problem where instances are single operational cycles, and the target to predict is the corresponding RUL value. This supervised ML regression task requires a dataset D of n observations/instances

$$D = \{x_i, y_i\}_{i=1}^{n}, \tag{4.1}$$

where $x_i \in R^{1\times p}$ is a vector of p variables with process and/or logistic information of an operational cycle, while y_i is the scalar with the RUL value for the corresponding instance [54]. For example, in the case of NASA datasets,

the p variables are the cycle, the three operational settings variables, and the 21 sensor measures. The n observations include the cycles of all time series in the dataset.

4.1.4 RUL Estimation for AutoML

In this chapter, the results of several experiments, performed on the NASA dataset, are exposed in detail in Section 4.3. Indeed, when dealing with RUL estimation, there are several issues that must be addressed. It is a regression task with the goal of predicting the target values as accurately as possible, but it differs from the classical problems of its family, since it requires some specific precautions. For instance, in many industrial scenarios, overestimating the RUL is worse than underestimating it, since a manufacturing company may prefer to waste a few operational cycles of its industrial equipment rather than having an unexpected machine breakdown. Therefore, when choosing the loss function to train the regressor, one has to take care of this peculiarity. Secondarily, it is clearly preferable to be precise in predicting the RUL towards the end of the equipment life rather than at the beginning, so more emphasis must be placed on the instances at the end of the time series of the given equipment. This can be done by weighting the instances differently, either with a custom loss or acting directly on the data. Moreover, the classical performance metrics adopted in regression, such as the Mean Absolute Error (MAE) and the Mean Squared Error (MSE), are not enough to evaluate the predictions of the RUL regressor, given the two issues outlined above. Better metrics, such as the Mean Absolute Percentage Error (MAPE), or even custom ones, should be adopted.

These are only some issues that characterise the Remaining Useful Life estimation. A Machine Learning expert must take all this into account when solving such a problem, and this applies equally or even more when developing an AutoML module. Therefore, the experiments performed try to tackle all these problems, while at the same time, try to identify a suitable algorithm, in particular in terms of low training time, for the AutoML library.

4.2 Dataset

NASA turbo engine datasets [52] are commonly used for RUL estimation and consist of multivariate time series, each one from a different engine. There

are four datasets with different characteristics, as visible in Figure 4.1, each divided into training and test sets. FD001, for instance, includes 100 training and 100 test time series of engines that operate in a single condition and with a single Fault Mode (HPC Degradation). The other folders involve more operative conditions and/or Fault Modes, but the experiments of this report are performed only on FD001 for simplicity. As described in the datasets, the engine operates normally at the beginning of its time series, and at some point, a fault occurs. The difference between training and test sets is that in the first the time series reach the engine failure, while in the latter they end at some point before the failure. The time units are operational cycles, and for each instance of the series 26 features are provided: the unit number, the cycle, 3 values that represent the operational settings, and 21 sensor measures.

Training Dataset		Testing Dataset				# of Conditions	Fault Mode
Dataset	Dimension	Dataset	Dimension	Dataset	Dimension	Engine	
FD001_train	20,631 × 26	FD001_test	13,096 × 26	RUL1	100 × 2	1	HPC Degradation
FD002_train	53,759 × 26	FD002_test	33,991 × 26	RUL2	259 × 2	6	HPC Degradation
FD003_train	24,720 × 26	FD003_test	16,596 × 26	RUL3	100 × 2	1	HPC & Fan Degradation
FD004_train	61,249 × 26	FD004_test	41,214 × 26	RUL4	248 × 2	6	HPC & Fan Degradation

Figure 4.1: Nasa turbo engine datasets

The series of these datasets are simulated with C-MAPSS (Commercial Modular Aero-Propulsion System Simulation), a tool to simulate a turbofan engine [52].

The experiments are mostly performed with Ceruleo, StatwolfML, Sklearn and Tensorflow libraries. Dataset is loaded using Ceruleo, that exploits a simple utility function to compute a new column with the Remaining Useful Life for each instance, using the cycle column.

4.3 Experiments

4.3.1 Random Forest vs Ridge

According to [67] [9] Random Forest (RF) is quite often adopted as a regressor for this task, so it is compared to the classical linear regressor Ridge. The K-Fold cross-validation grid search is performed to compare different hyperparameter configurations for the two models, with $k = 5$. For Ridge, three values of α,

i.e., the constant that multiplies the L2 regularisation term, are tested: 5, 10 and 15. For RF, five configurations of *n_estimators* and *max_depth* are tested, visible in Table 4.1. The metric adopted is the Mean Absolute Error (MAE) and the resulting best model is Random Forest with *max_depth=8* and *n_estimators=50*.

Random Forest	**n_estimators**	**max_depth**
Model 1	20	5
Model 2	20	8
Model 3	50	5
Model 4	50	8
Model 4	100	5

Table 4.1: Hyperparameters configurations for Random Forest (RF) .

The Mean Absolute Error (MAE) and Mean Squared Error (MSE) are widely used in the literature to evaluate the performance of the model for regression tasks and are defined in Equations 4.2 and 4.3, respectively:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \|y_i - \hat{y}_i\|, \tag{4.2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{4.3}$$

where $\hat{y}_i$ is the prediction of the regression model for the i-th instance x_i.

In Table 4.2, the best model according to the grid search is compared with the Ridge regressor with $\alpha = 15$ in terms of these metrics. The models are evaluated on 15 randomly selected time series from the FD001 test set.

Model	**MAE**	**MSE**
Random Forest (50, 8)	31.33	1840.74
Ridge (15)	32.91	1903.48

Table 4.2: Comparison of Random Forest with *max_depth=8* and *n_estimators=50* and Ridge with $\alpha = 15$ in terms of MAE and MSE.

More informative results are in the plots of Figures 4.2 and 4.3. In Figure 4.2, the predictions of the best estimator are compared with the true RULs of 15 engines, while Figure 4.3 also includes the Ridge predictions. Note that since these series are from the test set, most of them end much before the end of life of the engine.

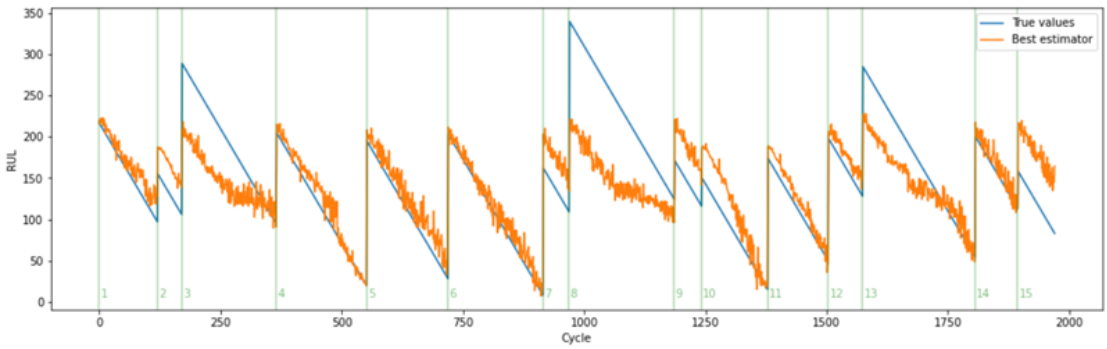


Figure 4.2: Predictions of RF on 15 time series.

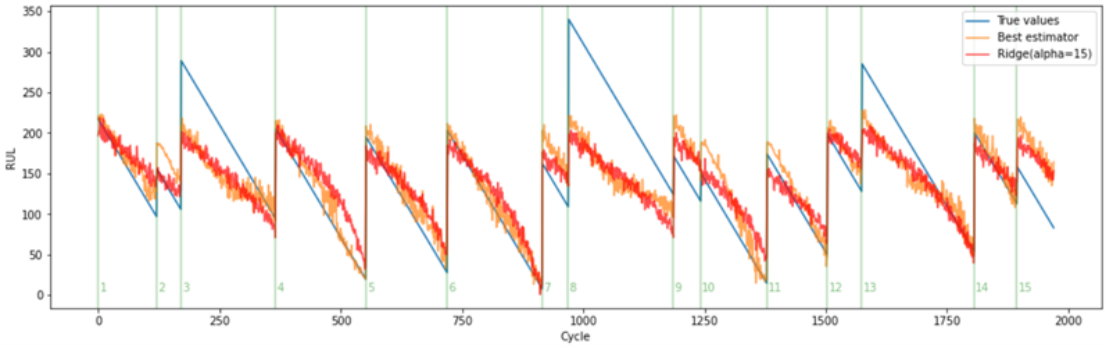


Figure 4.3: Comparison of RF and Ridge on 15 time series.

Consistent with the metric results of Table 4.2, RF seems to better fit the true RUL trends in most time series. This is more evident for series 4, 5 and 6, for which Ridge tends to overestimate the mid-life RULs.

However, both models show several critical issues. First, they mostly overestimate the target value. Since PdM aims to predict the failure of an industrial equipment allowing a company to intervene in time for maintenance, it is preferable to train a model with more pessimistic behaviour. As pointed out in much of the PdM literature [14][40][56][53], penalising overestimations of the target value more than underestimation is a safer approach for many maintenance scenarios. In particular for the NASA dataset, it is considerably worse to overestimate the RUL and run the risk of probable engine failures than to plan the repair too early, wasting engine life [14]. Nevertheless, there are also fewer life-threatening scenarios in which optimistic predictions are preferable [14]. A possible way to solve this issue is to train the regressor with an asymmetric loss function, as the one proposed in [14].

In view of the above, the Ridge regressor performs better for some engines, particularly the 3 and 13, for which RF at some point starts to overestimate the

RUL (see Figure 4.3).

The second point to consider is that not all instances of a time series have the same importance in PdM. Since the goal is to predict the RUL of an equipment to react in time, as far as we approach the end of life of the equipment, we may want to have more precise predictions to know exactly when to intervene. For instance, in engine 5 of Figure 4.2 RF overestimates the target toward the end of the series, while at the beginning the prediction was more precise. In engine 4, instead, there is the opposite trend, with a quite accurate regression for about the last 70 operational cycles.

Instance weighting can be used to force the model to focus more on the end of the series rather than the beginning.

The noisiness of predictions can be problematic, particularly near the end of life. Even if this issue may be easily solved with an appropriate post-processing, a correct hyperparameter tuning may alleviate it, as we will see for the learning rate of the gradient boosting machine.

Finally, engines 3, 8, and 13 show that both models tend to underestimate a lot the initial RUL of engines that have significantly longer than average life spans. This is probably due to the under-representation of these long-life engines in the training set, as proved by Figure 4.4: run-to-failure cycles (cycle in the sense of the entire engine life) with cycle durations between 250 and 350 are in the minority. In figure 4.5, similar considerations can be made for the test set.

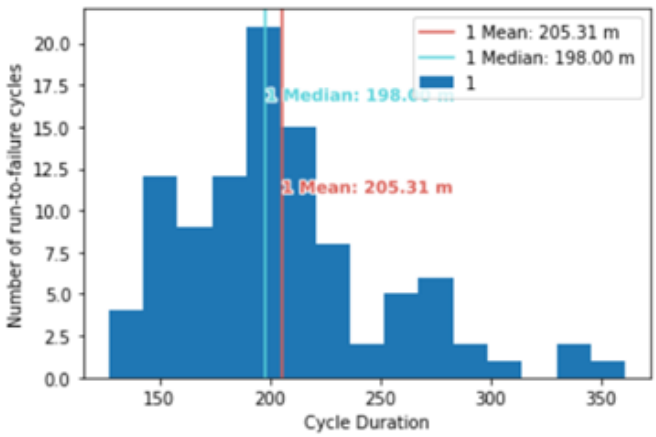


Figure 4.4: Train set engine life histogram.

Given the above considerations, it is possible to observe that the two metrics involved, MAE and MSE, are not enough to evaluate and compare models, as they do not consider all the issues of this task. For instance, they cannot

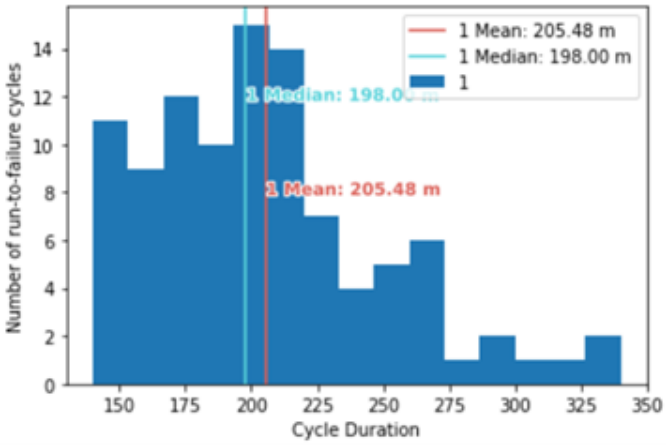


Figure 4.5: Test set engine life histogram.

penalise more overestimation than underestimation. Moreover, they give the same importance at all the instances of the time series, while we may prefer to penalise the errors in the end-of-life more. The MSE is also very sensitive to noisy predictions and to rare but large errors because of the quadratic term. For all of these reasons, other metrics will be considered in subsequent experiments.

4.3.2 Test with AutoML Library

Since the goal of these experiments is to find a good solution to the RUL estimation task for an AutoML library, an obvious approach may be to directly test the regression tools of the StatwolfML AutoML module on the NASA dataset. The best model provided as output is a CatBoost regressor with $n\_estimators = 1628$ and $max\_depth = 8$. The predictions are shown against the true values in Figure 4.6, considering the usual 15 engines from the test set.

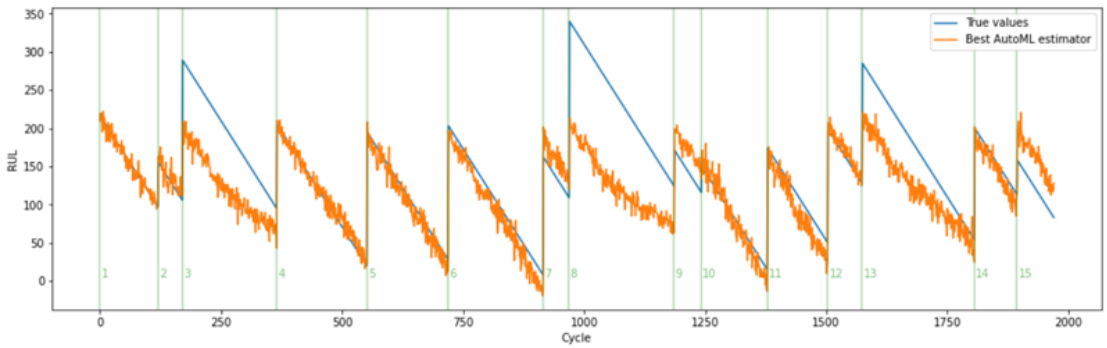


Figure 4.6: Predictions of the best estimator according to the AutoML module on the test set.

In Table 4.3, the performance of the best AutoML estimator is compared to that of the previous two estimators.

Model	**MAE**	**MSE**
Random Forest (50, 8)	31.33	1840.74
Ridge (15)	32.91	1903.48
CatBoost (1628, 8)	34.1	2446.5

Table 4.3: Performance of CatBoost compared with models of Table 4.2 .

Looking only at the MAE and the MSE, this model seems to perform worse than Random Forest and Ridge. However, this is not completely true, since CatBoost overestimates the RUL much less. For instance, RF mostly overestimates the targets of engine 6 (Figure 4.2), while CatBoost almost always underestimates them. The same holds true for engine 5, while for engines 2, 7, and 9 the error is reduced compared to RF. Figures 7 and 8 show the error distribution of the CatBoost and RF models on the test set, respectively, where the error is defined as the difference between the true value and the predicted value. Note how for CatBoost most of the error is positive, confirming the above considerations.

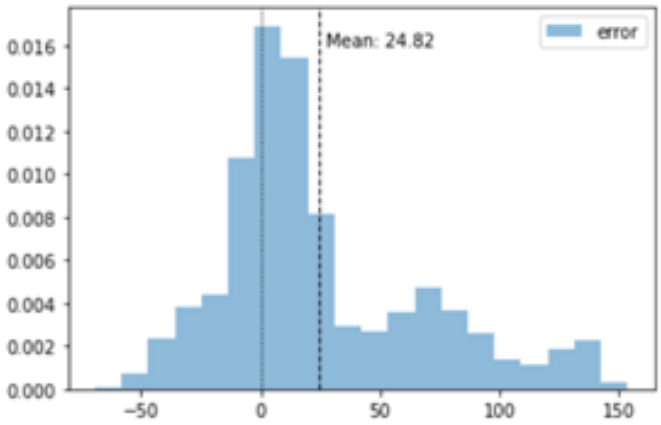


Figure 4.7: Catboost error distribution.

The major shortcomings of these results are the noise, in particular for series 11 and 13, if compared with the ones of RF, and, moreover, the fact that CatBoost tends to underestimate too much, often in the end-of-life of the engines, wasting life cycles (see engine 6).

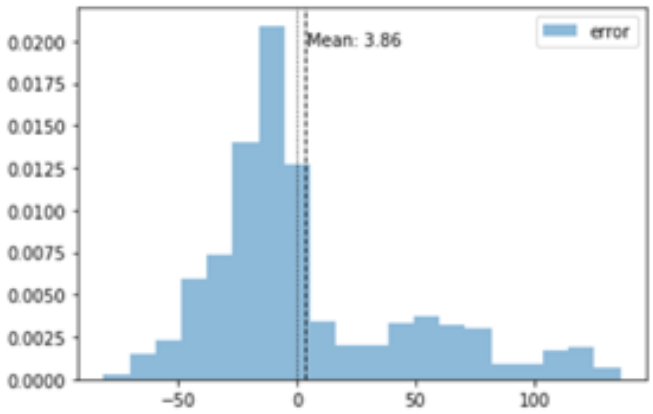


Figure 4.8: Random Forest error distribution.

4.3.3 Instance Weighting

To obtain better results, a possible way is to train regression models with custom loss functions, either modifying existing ones or creating them from scratch. However, there are few models that can be easily adapted to deal with such new losses. Among them, gradient boosting is the most intuitive. As described in the original work by J. H. Friedman [20], the gradient boosting machine can be used with any user-specified loss function, so it is well suited for those experiments.

The first attempted approach consists of modifying the MSE and involves instance weighting. By weighting the residual in the loss function, the weights can be included in the training process in a way that is independent of the model: $x = \omega(y, \hat{y})$ [14]. A common choice for ω consists of dividing the residual by the true value, providing the so-called relative residual of Equation 4.4:

$$\omega(y, \hat{y}) = \frac{y - \hat{y}}{\tilde{y}}, \tag{4.4}$$

where $\tilde{y}$ is y if $y \neq 0$ and some constant $0 < c < 1$ otherwise. By normalising residuals for the true RUL value, we give more importance to errors near the end of life rather than at the beginning. The MSE modified with these weights will be called the WMSE and is defined as follows.

$$WMSE = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{\tilde{y}_i} \right)^2, \tag{4.5}$$

where $\tilde{y}_i$ is defined as $\tilde{y}$ above.

Gradient Boosting with 100 trees (i.e., iterations), maximum depth of 8 and a learning rate of 0.08 is trained to minimise WMSE (about 22 seconds of training time). The value of the c constant for managing the zero denominator is set to 0.5. The predictions on the test set are in Figure 4.9.

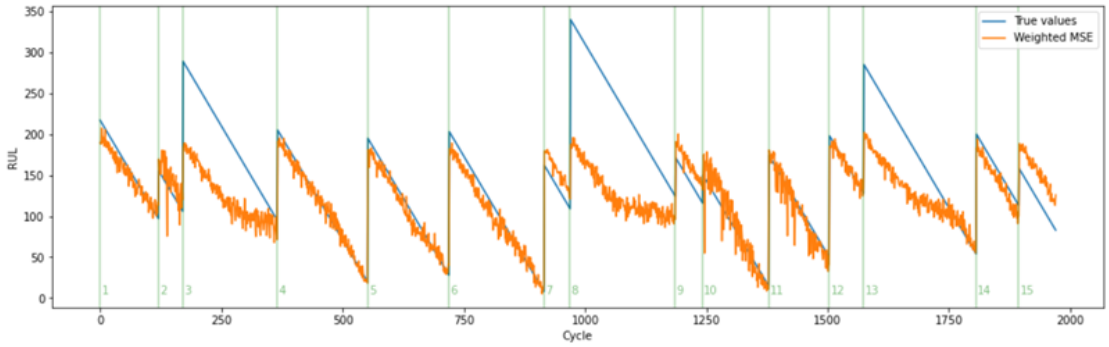


Figure 4.9: Predictions of the Gradient Boosting model with Weighted MSE loss function on test set.

To highlight the effect of weights in WMSE, the results of Figure 4.9 should be compared with those of Gradient Boosting trained with MSE, presented in Figure 4.14. In the last case, 150 iterations are performed, with a learning rate of 0.03 (the maximum depth is as before).

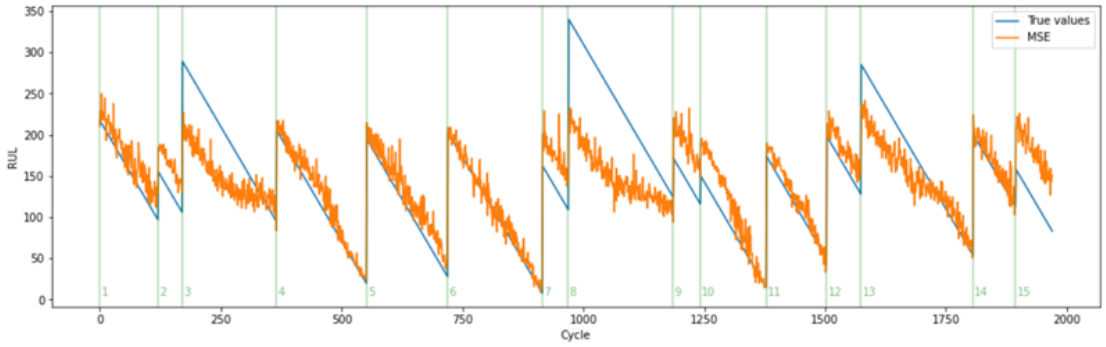


Figure 4.10: Predictions of the Gradient Boosting model with MSE loss function on test set.

The most noticeable difference is in terms of the noise of the predictions. Except for engines 2, 10, and 11, training the model with the weighted variant of MSE seems to greatly reduce the noise, providing more stable values. Adopting the relative residuals has a normalising effect on the model. Mathematically, this can be motivated by the working principle of Gradient Boosting that operates a kind of Gradient Descent. The model is updated at each iteration by training a

decision tree that corrects the errors of the previous iteration. These errors are computed from the negative gradient of the loss function with respect to each prediction. While the MSE gradient with respect to the residual is proportional to it, the WMSE gradient has a cubic term in the denominator that keeps it low:

$$\frac{\delta}{\delta\left(y_i - \hat{y}_i\right)} WMSE = 2 \times \frac{y_i^2 - \hat{y}_i{}^2}{y_i^3}. \tag{4.6}$$

Therefore, this term normalises the gradient, reducing consequently the error variance.

Reducing the learning rate has a similar normalisation effect. However, trying to reduce the learning rate with MSE loss does not work very well, providing less noisy but also much less precise predictions, especially in terms of RUL overestimation. Adopting a higher number of trees slightly compensates for this, at the cost of increased training time.

Another benefit of the weighted MSE is to provide a model that in many cases underestimates the RUL, especially at the end of the engine life.

An alternative way for weighting instances that do not involve the modification of the loss function consists of a subsampling of the training data that preserves more samples at the end of the useful life of the engines rather than in the beginning. In this way, we force the model to concentrate most on the end of the time series when a correct RUL prediction is critical.

A way to do this consists first of defining weights for the instances x_i:

$$W(x_i) = \frac{1}{\log\left(\max\left(y_i, 1\right)\right) + 1}. \tag{4.7}$$

These weights are then converted into probability for the sampling stage.

For each time series, only 60% of the instances are sampled without replacement, and are used to train Gradient Boosting with 150 trees, a learning rate of 0.03 and a maximum depth of 8. The predictions obtained optimising MSE loss are shown in Figure 4.11: as one might expect, the predictions are noisier and less precise at high values of the target, while they gain accuracy as we get closer to zero (engine 4, 5, 6).

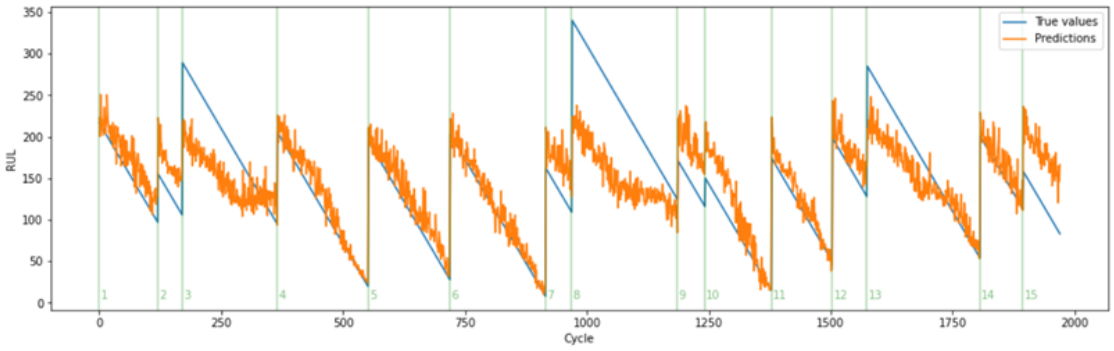


Figure 4.11: Predictions of the Gradient Boosting model with MSE loss function after subsampling.

4.3.4 Asymmetric Loss Function

Asymmetric loss functions are losses that penalise more overestimation of the target value than underestimation, or vice versa. In the literature several different regression asymmetric loss functions have been proposed [53], often developed for econometric modelling and forecasting [14]. A famous one is the so-called Linear Exponential (LINEX) loss [58], which is exponential on one side and approximately linear on the other side. Its main drawback is that it is controlled by only one parameter, so it is not possible to control the slope of the two portions independently. The Huber loss [27] is quadratic for small errors and linear otherwise, leading to a continuously differentiable function, quite similar to MAE but characterised by faster convergence. There also exist asymmetric variants of the MSE and MAE.

For these experiments, the customisable loss function presented in [14] is considered, due to its adaptability to various contexts. It is designed to ensure convexity and to be continuously differentiable. An implementation is also provided in the Ceruleo library.

Let us define the residual x as the difference between the target value y and the predicted value $\hat{y}$: $x = y - \hat{y}$. Then, the loss $\mathcal{L}(x)$ given the residual x is defined in a piecewise way:

$$\mathcal{L}(x) = \begin{cases} \ell_l(x), x < 0 \\ \ell_r(x), x \geq 0 \end{cases}, \tag{4.8}$$

where

$$\ell_r(x;\theta_r,\alpha_r,\psi_r) = \begin{cases} \alpha_r\theta_r\left(\theta_r + 2\psi_r\left(e^{\frac{x-\theta_r}{\psi_r}} - 1\right)\right), x \geq \theta_r \\ \alpha_r x^2, x < \theta_r \end{cases}, \tag{4.9}$$

$$\ell_l(x;\theta_l,\alpha_l,\psi_l) = \begin{cases} \alpha_l\left(-\theta_l\right)\left(-\theta_l + 2\psi_l\left(e^{\frac{x-\theta_l}{\psi_l}} - 1\right)\right), x \leq \theta_l \\ \alpha_l x^2, x > \theta_l \end{cases}. \tag{4.10}$$

Once the thresholds θ_r and θ_l are defined, the loss is quadratic in the range (θ_l, θ_r), while the behaviour is exponential otherwise. Another version of the loss adopts a linear function in place of the exponential, allowing us to derive the Huber loss [14].

As can be seen, 6 parameters control the loss shape, 3 for the left side $(\theta_l, \alpha_l, \psi_l)$ and 3 for the right side $(\theta_r, \alpha_r, \psi_r)$: by properly tuning them, it is possible to decide how much penalise one kind of error rather than another. For instance, ψ_r and ψ_l can be tuned to control the exponential growth. For these experiments, the adopted values are: $\theta_r = 5$, $\theta_l = -5$, $\alpha_r = 0.5$, $\alpha_l = 5$, $\psi_r = 20$ and $\psi_l = 20$. The plot of the loss as a value of the residual is shown in Figure 4.12 (in orange), and it is compared to MSE (in blue). Notice how, even for low errors, the overestimations (negative residuals) are much more penalised than underestimation (positive residuals). The presence of an exponential guarantees that large errors are strongly penalised in both cases.

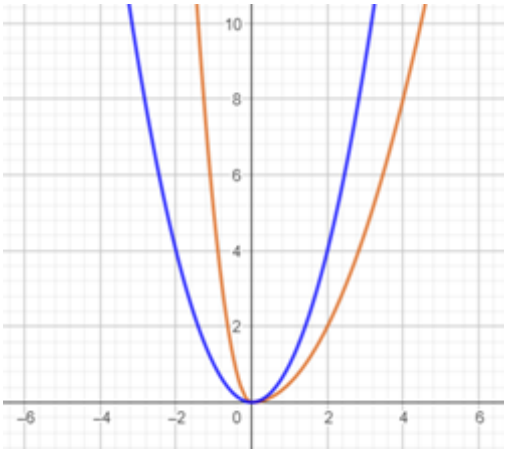


Figure 4.12: Custom Loss (in orange) compared to MSE (in blue) as a function of the residual.

Gradient Boosting Machine is the most suited algorithm for testing this

Custom Loss (CL). The hyperparameters are the following: $n\_trees = 50$, $learning\_rate = 0.1$, $max\_depth = 5$; note that the number of trees and the maximum depth are kept lower than in the previous cases since with this loss the training time is much longer (about 16 minutes against < 1 minute). Predictions are shown in Figure 4.13. In Figure 4.14, the results of GB with MSE loss are reproposed for comparison.

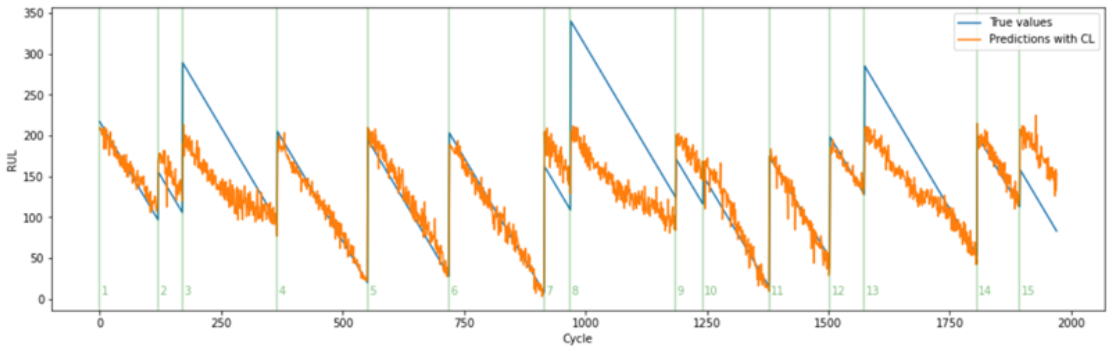


Figure 4.13: Predictions of the Gradient Boosting model with Custom Loss on the test set.

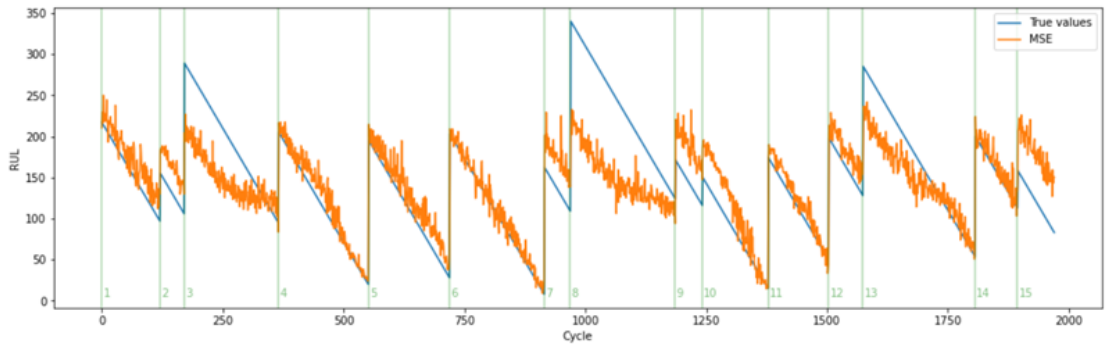


Figure 4.14: Predictions of the Gradient Boosting model with MSE on the test set.

In many cases, like in series 3, 6 and 13, the model trained with the CL tends to underestimate much more than the one trained with MSE. In others, the overestimation is strongly reduced (series 4, 5, 11 and 12). In general, predictions are less noisy. If we look at series 7 and 9 in Figure 4.14, we can see that several artefacts disappear in Figure 4.13. On the other hand, some noisy erroneous predictions appear in series 2, 10 and 11, but at least they are underestimations.

The CL-trained model is competitive with the WMSE-trained model, whose

results are shown in Figure 4.9. This latter is often effectively better in underestimate the RUL but at the price of a higher noisiness in predictions. However, they are both superior to the Random Forest of Figure 4.2 and to GB with MSE of Figure 4.14.

4.3.5 Experiment with a Multilayer Perceptron

The major drawback of adopting CL is the slow training time, a common problem when using Gradient Boosting. An alternative to this approach may be to train an Artificial Neural Network, like a Multilayer Perceptron, whose train directly involves a Gradient Descent.

The adopted model is characterised by one input layer of 25 nodes, two hidden layers of 20 and 5 nodes, respectively, and a final layer with a single node containing the predicted RUL. The activation functions are "ReLU" for the hidden layers and a linear function for the last one. The network summary with all parameters is shown in Figure 4.15.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1, 25)]           0
_________________________________________________________________
dense (Dense)                (None, 1, 20)             520
_________________________________________________________________
dense_1 (Dense)              (None, 1, 5)              105
_________________________________________________________________
dense_2 (Dense)              (None, 1, 1)              6
=================================================================
Total params: 631
Trainable params: 631
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.15: Multilayer Perceptron architecture.

The Multilayer Perceptron (MLP) is trained with Adam optimizers and a learning rate of 0.01. The maximum number of epochs is set to 25 and an early stopping procedure is adopted: the training is interrupted if there is no improvement of the validation loss for 3 consecutive epochs. In Figure 4.16 the predictions are shown along with the true values.

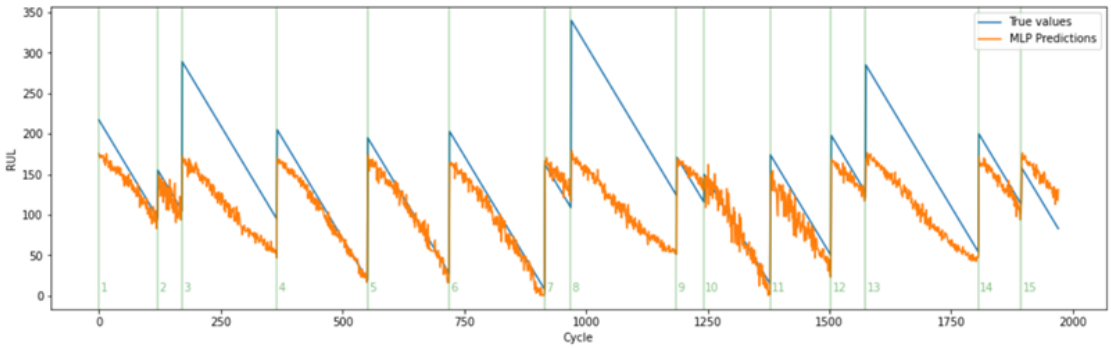


Figure 4.16: Predictions of the Multilayer Perceptron of Figure 4.15 with Custom Loss on the test set.

4.3.6 Metrics for Remaining Useful Lifetime Estimation

Up to now, most of the results analysis has been done by directly looking at the predictions of the models on the test set. Indeed, MAE and MSE show a series of limitations in measuring performance for this task, as seen previously. However, to make a more objective model comparison, it is necessary to define some metrics that capture and quantify the different critical aspects of RUL estimation that emerged during previous experiments.

A metric sometimes adopted in the PdM literature is the Mean Absolute Percentage Error (MAPE):

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i}|. \tag{4.11}$$

It is similar to the MAE, with the difference that the residuals are normalised by the true values. The main advantage of this metric is that usually returns a percentage value that is more easily interpretable than the mean of the absolute residuals. For this reason, this metric is also more suitable for comparing different models. Moreover, the presence of the target value at the denominator allows us to weight the errors differently, giving more importance to the residuals at the end of the series.

Another metric to evaluate the models must take into account the tendency of the models to underestimate rather than overestimate the RUL. To achieve this goal, define a confidence parameter ϵ. A first value that can be computed is the percentage of residuals that fall within the interval $(-\epsilon, \epsilon)$, intended as an acceptable range of errors, i.e., such that the errors within it do not differ too

much from the target both in terms of underestimation and overestimation. The value of ϵ is problem-specific, so it must be selected properly depending on the type of data at hand. Let us call this percentage $\%_\epsilon$.

Secondarily, define the area of underestimation $\mathcal{A}_u$ as the sum of the residuals of the cases of underestimations that fall outside the previous interval and similarly for the area of overestimation $\mathcal{A}_o$.

Then, a possible score is the following:

$$Score_{u-o} = \frac{\mathcal{A}_u - \mathcal{A}_o}{\mathcal{A}_u + \mathcal{A}_o}. \tag{4.12}$$

This score quantifies how much the model underestimates rather than overestimates and ranges from -1 to 1. The score is 1 if the model only underestimates (outside the range $(-\epsilon, \epsilon)$), and -1 otherwise. Values greater than 0 are clearly preferred if the task requires a more caution approach.

A drawback of this score is that it is not able to compare two models that both mostly underestimate. For instance, if we have a naïve model that always predicts zero RUL and a quite perfect model that always predicts the true value minus one, it is easy to see that both have $Score_{u-o} = 1$, since $\mathcal{A}_o = 0$. However, only the second model is good and for this reason it is preferable to use this score together with $\%_\epsilon$.

In Table 4, the three most interesting models seen up to now are compared with the baseline, that is, the model trained with MSE and Gradient Boosting, in terms of the previously described metrics, plus the MAE. The selected models are GB with WMSE, GB with CL and MLP with CL, whose predictions are in Figures 9, 13 and 15 respectively. The value of ϵ is set to 15 operational cycles.

Model	**MAPE**	**MAE**	$\%_\epsilon$	$\mathbf{Score_{u-o}}$
GB with WMSE	0.19	32.17	0.48	0.76
GB with CL	0.19	30.25	0.5	0.52
MLP with CL	0.24	39.83	0.38	0.89
GB with MSE	0.21	30.98	0.38	0.17

Table 4.4: Models comparison in terms of MAPE, MAE, $\%_\epsilon$ and $Score_{u-o}$.

The model that shows the best compromise in metric terms is probably the GB with WMSE. Indeed, it has the best combination of $\%_\epsilon$ and $Score_{u-o}$. MLP with CL is the approach that leads to the strongest underestimation, but at the price of having a relatively low number of residuals inside the range $(-\epsilon, \epsilon)$.

Consistently, its MAPE and MAE are the highest. Hopefully, by performing a proper hyperparameter tuning of the MLP, it is possible to achieve better results. The MAPE may be adopted as a metric to select the best architecture.

4.3.7 Experiments with Deep Neural Networks

Deep neural networks show remarkable results not only in computer vision, but also in time series forecasting tasks, like RUL prediction [66][51][26][28]: their capability to extract useful features at different levels of abstraction can be indeed exploited also in this field. However, providing as input to a DNN only a single instance may be too little informative content. For this reason, the time series are often divided into fixed-length windows, which are processed in their entirety by the network, which provides as output the predicted RUL for the last instance of the window.

For these experiments, both Convolutional and Recurrent NNs are tested. The architecture of the adopted CNN is visible in Figure 4.17, and is characterised at the feature extraction stage by three one-dimensional Convolutional layers, each with 64 neurons, followed by a Global max Pooling layer. The activation function is the ReLu for the first three layers. After the pooling, two final Fully Connected layers are responsible for preforming the regression and have 64 and 1 neurons, respectively. The last two activation functions are ReLU and the linear one. The total number of parameters is 33,793.

Other architectures, obtained by slightly modifying the number of layers/neurons of the Convolutional layers, have been tested and evaluated in terms of MAPE, as visible in Table 4.5. However, the most promising architecture remains the one with 3 layers of 64 neurons in Figure 4.17.

Model	**MAPE**
2l, 64n	0.23
2l, 32n	0.25
3l, 64n	0.21
3l, 32n	0.22
4l, 64n	0.22

Table 4.5: Comparison of different CNN architectures (l=number of convolutional layers, n=number of neurons) .

The size of the time series window is selected similarly: the window of length

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 8, 25)]           0
_________________________________________________________________
conv1d (Conv1D)              (None, 8, 64)             4864
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 8, 64)             12352
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 8, 64)             12352
_________________________________________________________________
global_max_pooling1d (Global (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 33,793
Trainable params: 33,793
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.17: Convolutional Neural Network architecture.

8 shows the best MAPE in Table 4.6. So, all the series are divided into windows of size 8 with a step of 1, in such a way as to obtain a prediction to almost all instances.

Window size	**MAPE**
32	0.26
16	0.24
8	0.21

Table 4.6: Performance of model of Figure 4.17 with different window sizes.

CNN predictions with the Custom Loss in the test set are shown in Figure 4.18. The behaviour is quite similar to the one with the MLP of Figure 4.16, with a large underestimation at the beginning of the series (see engines 4, 5, 6), while the model gains precision towards the end (engines 5 and 10). Predictions are less noisy, probably due to the window-based adopted approach that helps to stabilise predictions. This is particularly evident for series 10 and 11, for which all previous tested models show some difficulties. However, the models also show some problems. For instance, it seems to treat series 1 and 2 as one, while in series 13 the predictions do not converge to the real ones, unlike the MLP. In general, towards the end it is less precise than previously seen models.

Among the different kind of Deep NNs, the Recurrent ones seem to be the

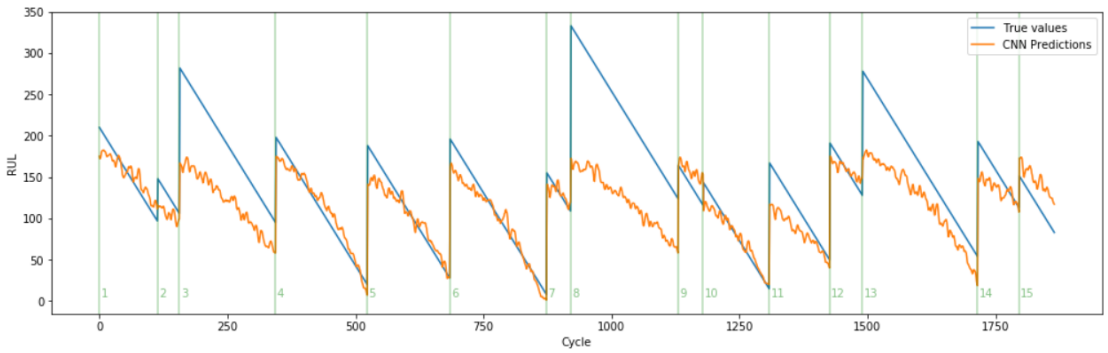


Figure 4.18: Predictions of the CNN of Figure 4.17 with Custom Loss on the test set.

most suited for dealing with time series, where instances are characterised by temporal relations. Indeed, RNNs operate as they had an internal memory that keeps information about previously seen instances [26]; this information is then used together with the features of the next instance to make the prediction. The main RNN drawback is the gradient vanishing. For this reason, the long short-term memory (LSTM) is often adopted as a valid alternative: it uses three gates (input, output and forget gates) to add or remove information from a memory cell, mitigating the gradient vanishing issue[26].

In the literature, LSTMs are widely used for RUL prediction, so a simple LSTM-based network is also proposed in these experiments. It is characterised by a single LSTM layer with 64 units, followed by a Global max Pooling, and two Fully Connected layers, with 64 and 1 nodes, respectively. The activation functions for the last layers are ReLU and linear, respectively. The parameters are 27,265, as visible in Figure 4.19.

Figure 4.20, instead, shows the model predictions on the test set. The trend that seems shared by all Neural Networks models involves an initial large underestimation, which, however, is not a big deal since it is not strictly necessary to be precise at the beginning of the series. The LSTM predictions are, however, more accurate than the CNN ones, in particular close to the end of engine life (see engines 4, 5, and 6). It is capable of capturing the trends of series 1 and 2 and converge to the true values on series 13. For engines 10 and 11, the results remain much better than the one of the MLP of Figure 4.16. Series 9 and 15 are still overestimated.

For this last model, it is interesting to see the predictions also on the validation set (Figure 4.21). In this case, all series reach zero and it is possible to better

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 8, 25)]           0
_________________________________________________________________
lstm (LSTM)                  (None, 8, 64)             23040
_________________________________________________________________
global_max_pooling1d (Global (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 27,265
Trainable params: 27,265
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.19: Recurrent Neural Network architecture.

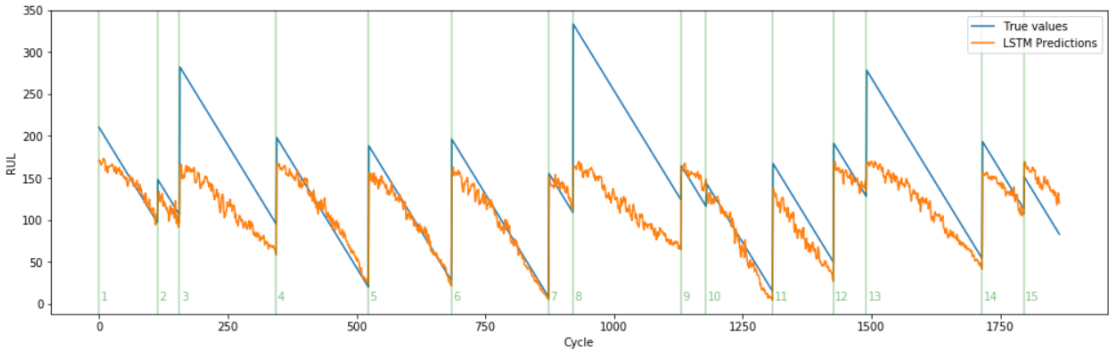


Figure 4.20: Predictions of the LSTM model with Custom Loss on the test set.

observe the precision of the model toward the end of the engine life: in almost all cases the predictions are accurate and, in general, there are no cases of severe overestimation close to zero. Except for the long-life engines 1, 5, and 10 and the strange behaviour of engine 4, the predictions match quite well with the true values.

The performance of the two Deep Learning models of this subsection in terms of the metrics presented are compared in Table 4.7 with previous methods. Note that GB with MSE is again the baseline.

As can be guessed from Figures 4.18 and 4.20, the CNN and RNN models have a high value of $Score_{u-o}$ and a comparable percentage of predictions within the range defined by ϵ. The most promising approach is probably the one with the LSTM model. Indeed, it has one of the best combinations of $Score_{u-o}$ and $\%_{\epsilon}$, similar to that with MLP, but with lower MAPE and MAE.

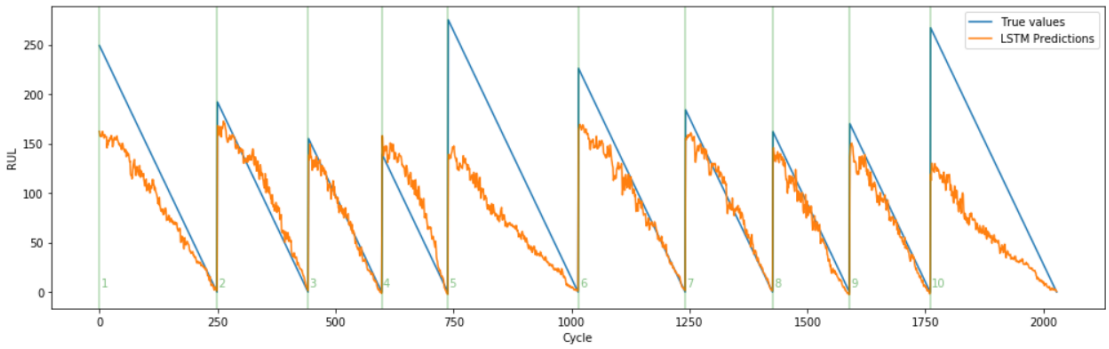


Figure 4.21: Predictions of the LSTM model with Custom Loss on the validation set.

Model	**MAPE**	**MAE**	$\%_\epsilon$	$\textbf{Score}_{\textbf{u-o}}$
GB with WMSE	0.19	32.17	0.48	0.76
GB with CL	0.19	30.25	0.5	0.52
MLP with CL	0.24	39.83	0.38	0.89
CNN with CL	0.21	33.94	0.41	0.79
LSTM with CL	0.22	36.46	0.39	0.87
GB with MSE	0.21	30.98	0.38	0.17

Table 4.7: Models comparison in terms of MAPE, MAE, $\%_\epsilon$ and $Score_{u-o}$.

4.3.8 Interpretability with LIME

The focus on results' interpretability is critical not only for Anomaly Detection, but also for the RUL estimation task. Being able to justify the predictions and provide evidence about which features are more crucial in determining the model output is an important information to provide to the final user.

Local Interpretable Model-Agnostic Explanations (LIME) [48] is a software package that can identify an interpretable model, locally faithful to the classifier/regressor whose predictions we aim to explain [48]. The interpretations are local since they are provided for single predictions.

As described in the original paper, the working principle of LIME is based on the trade-off between the (local) fidelity of the interpretable model to the original model and its interpretability. Let G be the class of interpretable models, i.e., that can be easily understood by the user by means of textual/graphical representations (for instance, random forests or linear models are of this kind). We can define $\Omega(g)$ as a measure of the complexity (in the sense of less interpretability) of a $g \in G$. In LIME, Ω is the class of sparse linear models and $\Omega(g)$

is the number of nonzero weights.

The model we want to explain can be called $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Given an instance x, define $\pi_x(z)$ as a proximity measure of x to z. $\pi_x(z)$ is used to define a locality around x. Then, $\mathcal{L}(f, g, \pi_x(z))$ is used to quantify how unfaithful is the approximation g with respect to f in the locality defined by $\pi_x(z)$.

Therefore, LIME provides an explanation for the instance x, called $\xi(x)$, facing the trade-off between fidelity and interpretability:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x(z)) + \Omega(g) \tag{4.13}$$

If $x \in \mathbb{R}^d$ is the original representation of the instance, its interpretable representation can be defined as $x' \in \{0,1\}^{d'}$ where 0 and 1 represent the absence or the presence of the so-called interpretable components. To learn the local behaviour of f in a way that is model-agnostic, LIME sample instances around x' by randomly drawing the non-zero components of x'. Given a new instance $z' \in \{0,1\}^{d'}$, it is possible to return to its original representation z to obtain the prediction $f(z)$. Finally, the dataset of perturbed samples $\mathcal{Z}$ (including labels), with the instances weighted with $\pi_x(z)$, is used to derive the explanation $\xi(x)$ [48].

The model considered for interpretability analysis in this section is the LSTM one. In Figure 4.23, there is an example of an explanation for instance 770 of the test set (Figure 4.22). It is a window from series 6, with a true RUL of 111 and predicted one of 107.

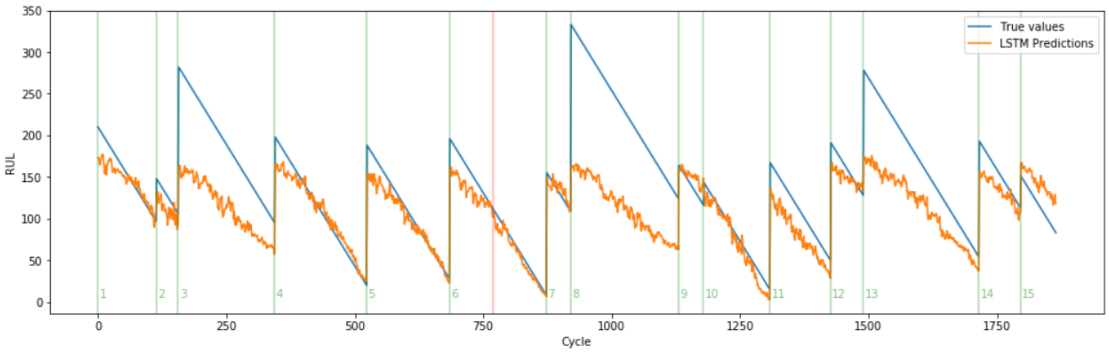


Figure 4.22: Predictions of the LSTM model with Custom Loss on the test set, with highlighted instance 770 (red vertical line).

On the left of Figure 4.23 is specified the value predicted by the model (107.20) together with the minimum and maximum values for the instance. On

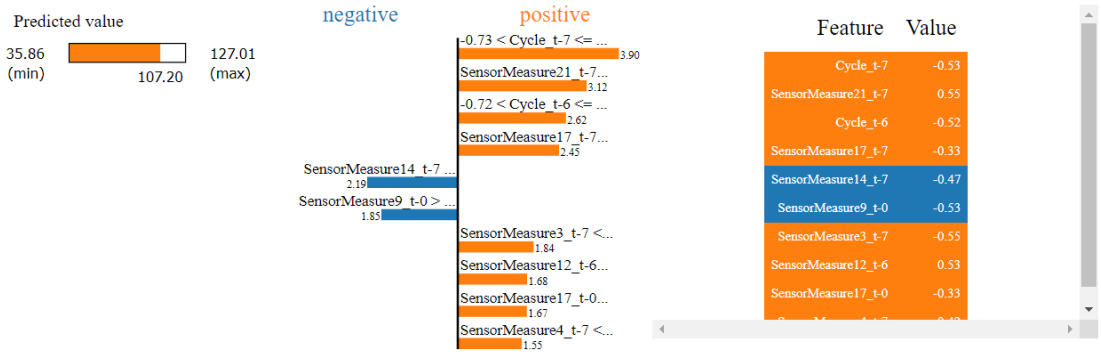


Figure 4.23: Explanation with LIME of instance 770 of the test set.

the right, instead, there is the list of values (normalised between -1 and 1) of the 10 most important features of the window. Concatenated with the feature name is indicated the instance to which they are related, so, for instance, "Cycle_t-7" is the value of the "Cycle" feature for the last window instance. The most interesting information is in the middle of the figure, where the 10 most relevant features are sorted by their weights, which quantify how much they contribute to the prediction. In particular, a range for the features is displayed. Each feature can impact positively, i.e., increasing the RUL value, or negatively, decreasing it.

In this example, the most prominent feature is the cycle of the last instance, with a weight of 3.9. "Cycle_t-7", whose value is -0.53 and falls in the range (-0.73, -0.45), impacts the prediction positively. Only two features decrease the value, which are "SensorMeasure14_t-7" and "SensorMeasure0_t-0".

A more interesting usage of LIME is described in the following. Let us consider the results of the model on the validation set (Figure 4.21). As already observed, series 4 has a strange behaviour if compared, for instance, with the previous series 3. Indeed, for most engine lifetime, the model highly overestimates the RUL, correcting the predictions only for the last 20/30 instances. By comparing the explanations of two instances with the same true value but from these two different series, may be possible to investigate the reasons behind the erroneous predictions of series 4. A comparison of this kind should, however, be done only with series with similar length, otherwise the feature related to cycles may mislead the explanations (in general, it impacts a lot on the prediction, like the previous example).

The two selected windows are 540 and 680, from series 3 and 4, respectively (see Figure 4.24). The true RUL is 58 for both, while the predicted is 59 for series

3 (low overestimation) and 87 for series 4 (severe overestimation).

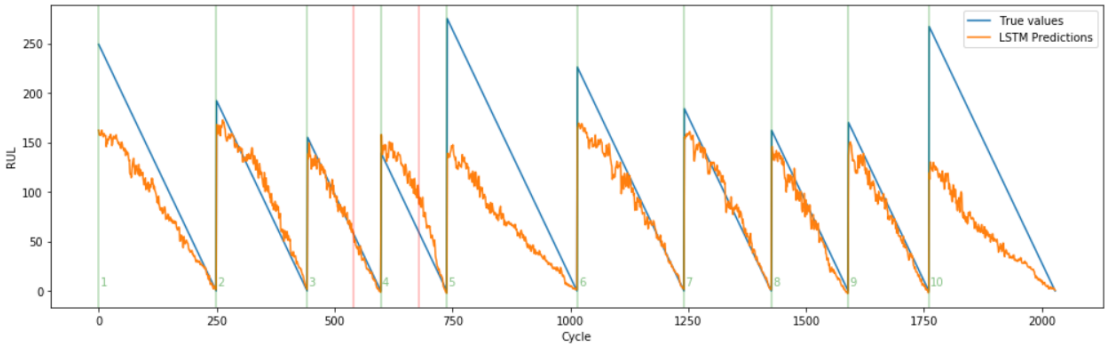


Figure 4.24: Predictions of the LSTM model with Custom Loss on the validation set, with highlighted instances 540 and 680 (red vertical lines).

The explanations are in Figures 4.25 for window 540 and 4.26 for window 680. As expected, in the case of low overestimation, there are more important features impacting in a negative way with respect to the instance of series 4 (6 against 3). Moreover, the four instances that impact positively on window 540 coincide with the first four of window 680. The most obvious difference is the absence of "SensorMeasure20_t-7" and "SensorMeasure21_t-7" in the features of window 680. These contribute a lot to reducing the predicted RUL for window 540, so their absence may explain the issue of series 4.

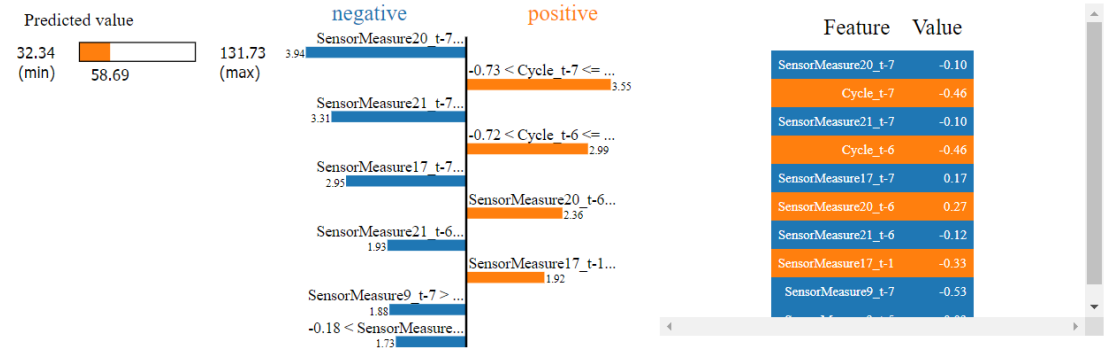


Figure 4.25: Explanation with LIME of instance 540 of the validation set.

Figure 4.27 shows the distributions of the feature "SensorMeasure20" for series 3 and series 4, which seem to show some differences. The meaning of this sensor measure can be obtained in [41], where it is described as "High-pressure turbines Cool air flow".

However, this information is not enough to infer if the explanation provided by LIME refers to some real physical phenomena or if it shows only a spurious

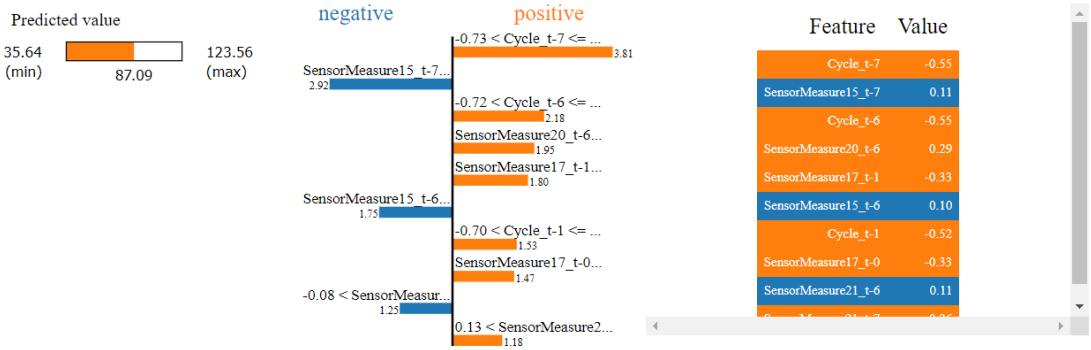


Figure 4.26: Explanation with LIME of instance 680 of the validation set.

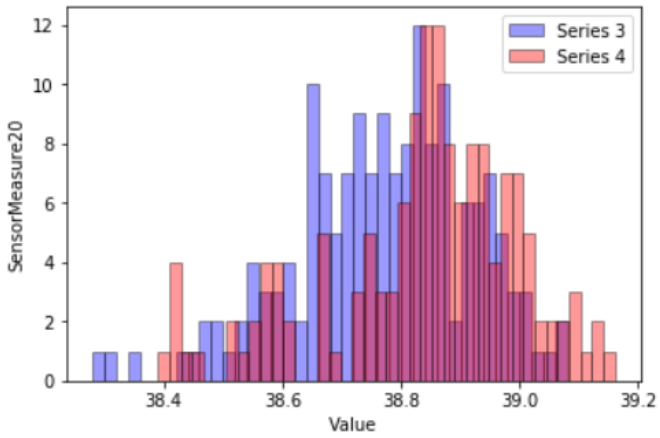


Figure 4.27: Histogram of distribution of "SensorMeasure20" for series 3 (in blue) and series 4 (in red).

correlation captured by the LSTM model. So, this kind of analysis may be used as a basis for leading the discussion with domain experts.

4.3.9 Conclusions

Other more sophisticated models have been tested. For instance, the approach proposed in "Temporal Convolutional Memory Networks for Remaining Useful Life Estimation of Industrial Machinery" [28], combines three one-dimensional Convolutional layers, each one followed by a max Pooling layer, with two LSTMs, combining the advantages of the CNNs with the ones of the RNNs. However, the results on the FD001 dataset are not as good and have not been proposed here. The adopted implementation can be found in the Ceruleo library. In "XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification" [18], a CNN with two branches is exploited

to extract both temporal features and features related to input variables. The branches outputs are then concatenated and provided to a Global Average Pooling. Originally developed for time series classification, in Ceruleo the model has been adapted to regression tasks. An interesting feature is the model exaplainability by means of Grad-CAM. Also in this case, the results are not comparable with the best models seen previously.

In general, the experiments performed in this Section seem to show that more "classical" approaches often work better. Indeed, in Table 4.7, we can see that the two models trained with Gradient Boosting have the lowest MAPE and MAE (if we exclude GB with MSE). The best value for $\%_\epsilon$ is always of GB with CL, which however has the lowest value $Score_{u-o}$, after the one of GB with MSE. Among the best combinations of $\%_\epsilon$ and $Score_{u-o}$ there is probably GB with WMSE, even if NN-based approaches also show good results (the best $Score_{u-o}$ is of MLP with CL). Taking into account also the training times, the GB models are surprisingly faster than Neural Network ones, and in particular, the training time for GB with WMSE is below one minute.

More attempts can be done toward the results' interpretability. Indeed, the analysis performed in the previous Subsection may be difficult to understand by a user with low or no experience in Machine Learning. The development of interpretability solutions specifically designed for Predictive Maintenance is definitely a very valuable research direction.

5

Conclusions and Future Work

In this last chapter, the main results are summarised and some directions for future work are outlined, with a view to improving the proposed solutions.

For the Anomaly Detection task on tabular data, MetaOD presents the most methodological solution to the CASH problem, finding, among several different combinations of algorithm and hyper-parameter configurations, the most suited for the dataset at hand. This is achieved through meta-learning, a technique that exploits prior knowledge on similar datasets to find the best solution for a new dataset. This allows us to solve in a smart way the CASH problem that otherwise would be unsolvable for the unsupervised version of the AD task. This tool was tested on two real datasets and a synthetic one, showing results comparable to those of a well-established algorithm for Outlier Detection, that is, the Isolation Forest. The main drawback of MetaOD lies in the assumption that similarity between datasets is enough to determine the most suitable model. However, the multifaceted nature of the anomalies and the different domains from which they came from makes the MetaOD assumption vulnerable. A possible way to strengthen this AutoML solution consists in comparing the distances of the new dataset and the historical ones with the distribution of distances of the historical datasets, to see if the new data are close enough to them. These distances can be computed by considering the meta-feature vectors associated with the datasets. Then, given a threshold, if the data are too far from the distribution, the Isolation Forest can be used.

To sum up, MetaOD is a valid solution to the CASH problem for unsupervised Anomaly Detection and can be used in an AutoML tool. However, further

work must be done to improve this approach by mitigating the negative effects behind MetaOD's assumptions. Moreover, the integration of the AcME interpretability with MetaOD is essential for providing user and Data Scientist insight about the impact of individual features in the instance abnormality, making the AutoML tool more transparent and reliable.

The work done for the Visual Anomaly Detection task has been set up differently. Its unsupervised nature and the absence, in literature, of solutions similar to MetaOD, make it impossible to solve the CASH problem for the task and forced the research in identifying a good state-of-the-art model that satisfies several AutoML-related constraints. Interpretability of the results, training time, performance, robustness to training data numerosity are the main criteria adopted in the comparison of eight well-established approaches to Visual AD. Padim and Patchcore are the best. Padim is the fastest in training, but it shows vulnerabilities when dealing with non-aligned dataset, with noticeable drops in performance. Patchcore has the best performance at the price of a slowdown in the training phase. These two models can be used alternatively, depending on the characteristics of the dataset and the time requirements. However, a possible future work could focus on adopting Computer Vision technique to realign not aligned dataset as a pre-processing step. Once this is achieved, Padim would be a sufficiently powerful model for a wider range of datasets, providing the results also quickly.

Experiments on the Remaining Useful Life estimation task were characterised by the testing of numerous algorithms and models, the adoption of custom loss functions, and the introduction of new metrics for the evaluation of predictions. Indeed, as outlined in the previous chapter, there are many critical aspects that must be considered in this task before developing an AutoML solution. The results show that classical approaches, such as using Gradient Boosting to train a model with the WMSE, often yield better results, in terms of the combination of the adopted metrics, than more advanced ones. The approaches that can be found in the literature, despite often being sophisticated, usually work well only on specific datasets. Therefore, they are not suitable as AutoML tools, which have to interface with even very different datasets.

The adoption of custom loss functions, in addition to being instrumental in obtaining a good model, can also be useful in providing the user of the AutoML tool with a customisation option. The asymmetric CL adopted in most experiments is characterised by several parameters that allow us to define

the shape of the quadratics and exponentials components, in such a way as to penalise less or more RUL overestimation and underestimation. The AutoML tool may allow the user to decide whether to penalise RUL overestimates by a lot or a little, and this choice can be reflected in the loss function because of its versatility.

The experiments were performed in the FD001 folder of the NASA datasets, which includes only one operative condition and a single fault mode. However, in most scenarios, this is not the case. Therefore, future work will include the development of AutoML solutions that take into account the possibility that the equipment may operate in different conditions and that there may be different types of failure.

Finally, the LIME interpretability can be used as a diagnostic tool to help domain experts in assessing which features of the industrial equipment are responsible for the breakdown, and to assess if the model really reflects some characteristic of the machine, or if it captures only spurious correlations. The use of this tool can be further explored in future work.

In general, AutoML is a promising tool in democratising Machine Learning, making this amazing field accessible to more and more people and companies, reducing the costs and knowledge required to approach it. This work shows how AutoML can be a viable solution also for some manufacturing tasks, falling within the Industry 4.0 paradigm, such as Anomaly Detection, Visual Anomaly Detection, and Remaining Useful Life estimation. However, research in AutoML for these tasks is still in its infancy, and we can expect great progress in the future, for instance in solving in a more methodological way the CASH problem for the Visual AD task, as done for the AD task. In any case, when developing AutoML solutions, it is important to remember that the objective is to make the ML accessible to non-experts, so the focus must always be on aspects such as the interpretability of models and the user-friendliness of the tool. Today, the manufacturing industry is increasingly discovering the importance of extracting valuable information from data, and machine learning plays a vital role in this. Therefore, it can be expected that AutoML-based solutions will acquire great value as the Industry 4.0 paradigm spreads among manufacturing companies.

References

[1] Mounia Achouch et al. "On predictive maintenance in industry 4.0: Overview, models, and challenges". In: *Applied Sciences* 12.16 (2022), p. 8081.

[2] Nilesh A Ahuja et al. "Probabilistic modeling of deep features for out-of-distribution and adversarial detection". In: *arXiv preprint arXiv:1909.11786* (2019).

[3] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. "Ganomaly: Semi-supervised anomaly detection via adversarial training". In: *Asian conference on computer vision*. Springer. 2018, pp. 622–637.

[4] Samet Akcay et al. "Anomalib: A Deep Learning Library for Anomaly Detection". In: *arXiv preprint arXiv:2202.08341* (2022).

[5] Maroua Bahri et al. "AutoML: state of the art with a focus on anomaly detection, challenges, and research directions". In: *International Journal of Data Science and Analytics* (2022), pp. 1–14.

[6] Paul Bergmann et al. "MVTec AD–A comprehensive real-world dataset for unsupervised anomaly detection". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 9592–9600.

[7] Oleg Bezrukavnikov and Rhema Linder. "A neophyte with automl: Evaluating the promises of automatic machine learning tools". In: *arXiv preprint arXiv:2101.05840* (2021).

[8] Alexander Buslaev et al. "Albumentations: fast and flexible image augmentations". In: *Information* 11.2 (2020), p. 125.

[9] Thyago P Carvalho et al. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (2019), p. 106024.

[10] David Dandolo et al. "AcME—Accelerated model-agnostic explanations: Fast whitening of the machine-learning black box". In: *Expert Systems with Applications* 214 (2023), p. 119115.

[11] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[12] Thomas Defard et al. "Padim: a patch distribution modeling framework for anomaly detection and localization". In: *International Conference on Pattern Recognition*. Springer. 2021, pp. 475–489.

[13] Kunyuan Deng et al. "A remaining useful life prediction method with long-short term feature processing for aircraft engines". In: *Applied Soft Computing* 93 (2020), p. 106344.

[14] Lukas Ehrig et al. "Customizable Asymmetric Loss Functions for Machine Learning-based Predictive Maintenance". In: *2020 8th International Conference on Condition Monitoring and Diagnosis (CMD)*. IEEE. 2020, pp. 250–253.

[15] Moustafa Elnadi and Yasser Omar Abdallah. "Industry 4.0: critical investigations and synthesis of key findings". In: *Management Review Quarterly* (2023), pp. 1–34.

[16] Andrew Emmott et al. "Anomaly detection meta-analysis benchmarks". In: *arXiv preprint ArXiv:1503.01158* (2016).

[17] Nick Erickson et al. "Autogluon-tabular: Robust and accurate automl for structured data". In: *arXiv preprint arXiv:2003.06505* (2020).

[18] Kevin Fauvel et al. "Xcm: An explainable convolutional neural network for multivariate time series classification". In: *Mathematics* 9.23 (2021), p. 3137.

[19] Matthias Feurer et al. "Efficient and robust automated machine learning". In: *Advances in neural information processing systems* 28 (2015).

[20] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[21] Morteza Ghobakhloo. "Industry 4.0, digitization, and opportunities for sustainability". In: *Journal of cleaner production* 252 (2020), p. 119869.

[22] Morteza Ghobakhloo. "The future of manufacturing industry: a strategic roadmap toward Industry 4.0". In: *Journal of manufacturing technology management* (2018).

[23] Alasdair Gilchrist. *Industry 4.0: the industrial internet of things*. Springer, 2016.

[24] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[25] Denis Gudovskiy, Shun Ishizaka, and Kazuki Kozuka. "Cflow-ad: Real-time unsupervised anomaly detection with localization via conditional normalizing flows". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 98–107.

[26] Che-Sheng Hsu and Jehn-Ruey Jiang. "Remaining useful life estimation using long short-term memory deep learning". In: *2018 ieee international conference on applied system invention (icasi)*. IEEE. 2018, pp. 58–61.

[27] Peter J Huber. "Robust estimation of a location parameter". In: *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[28] Lahiru Jayasinghe et al. "Temporal convolutional memory networks for remaining useful life estimation of industrial machinery". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2019, pp. 915–920.

[29] Nasser Jazdi. "Cyber physical systems in the context of Industry 4.0". In: *2014 IEEE international conference on automation, quality and testing, robotics*. IEEE. 2014, pp. 1–4.

[30] Haifeng Jin, Qingquan Song, and Xia Hu. "Auto-keras: An efficient neural architecture search system". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1946–1956.

[31] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. "Remaining useful life (RUL) prediction of equipment in production lines using artificial neural networks". In: *Sensors* 21.3 (2021), p. 932.

[32] Lars Kotthoff et al. "Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA". In: *Automated machine learning*. Springer, Cham, 2019, pp. 81–95.

[33] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. "Angle-based outlier detection in high-dimensional data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452.

[34] Kwei-Herng Lai et al. "Tods: An automated time series outlier detection system". In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 35. 18. 2021, pp. 16060–16062.

[35] Heiner Lasi et al. "Industry 4.0". In: *Business & information systems engineering* 6.4 (2014), pp. 239–242.

[36] Yuening Li et al. "Automated Anomaly Detection via Curiosity-Guided Search and Self-Imitation Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[37] Yuening Li et al. "Pyodds: An end-to-end outlier detection system with automated machine learning". In: *Companion Proceedings of the Web Conference 2020*. 2020, pp. 153–157.

[38] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[39] Randal S Olson and Jason H Moore. "TPOT: A tree-based pipeline optimization tool for automating machine learning". In: *Workshop on automatic machine learning*. PMLR. 2016, pp. 66–74.

[40] Ingeborg de Pater and Mihaela Mitici. "Novel metrics to evaluate probabilistic remaining useful life prognostics with applications to turbofan engines". In: *PHM Society European Conference*. Vol. 7. 1. 2022, pp. 96–109.

[41] Cheng Peng et al. "A Remaining Useful Life prognosis of turbofan engine using temporal and spatial feature fusion". In: *Sensors* 21.2 (2021), p. 418.

[42] Ana C Pereira and Fernando Romero. "A review of the meanings and the implications of the Industry 4.0 concept". In: *Procedia Manufacturing* 13 (2017), pp. 1206–1214.

[43] Juan-Carlos Perez-Cortes et al. "A System for In-Line 3D Inspection without Hidden Surfaces". In: *Sensors* 18.9 (2018), p. 2993.

[44] Tomáš Pevnỳ. "Loda: Lightweight on-line detector of anomalies". In: *Machine Learning* 102.2 (2016), pp. 275–304.

[45] Wenwen Qi, Chong Xu, and Xiwei Xu. “AutoGluon: A revolutionary framework for landslide hazard analysis”. In: *Natural Hazards Research* 1.3 (2021), pp. 103–108.

[46] Rahul Rai et al. “Machine learning in manufacturing and industry 4.0 applications”. In: *International Journal of Production Research* 59.16 (2021), pp. 4773–4778.

[47] Danilo Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[48] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[49] Karsten Roth et al. “Towards total recall in industrial anomaly detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 14318–14328.

[50] Takaya Saito and Marc Rehmsmeier. “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets”. In: *PloS one* 10.3 (2015), e0118432.

[51] Giduthuri Sateesh Babu, Peilin Zhao, and Xiao-Li Li. “Deep convolutional neural network based regression approach for estimation of remaining useful life”. In: *International conference on database systems for advanced applications*. Springer. 2016, pp. 214–228.

[52] Abhinav Saxena et al. “Damage propagation modeling for aircraft engine run-to-failure simulation”. In: *2008 international conference on prognostics and health management*. IEEE. 2008, pp. 1–9.

[53] Anıbal Silva, Rita P Ribeiro, and Nuno Moniz. “Model Optimization in Imbalanced Regression”. In: *International Conference on Discovery Science*. Springer. 2022, pp. 3–21.

[54] Gian Antonio Susto et al. “Machine learning for predictive maintenance: A multiple classifier approach”. In: *IEEE transactions on industrial informatics* 11.3 (2014), pp. 812–820.

[55] Xian Tao et al. “Deep learning for unsupervised anomaly localization in industrial images: A survey”. In: *IEEE Transactions on Instrumentation and Measurement* (2022).

[56] Andrei Tolstikov, Frederik Janssen, and Johannes Fürnkranz. “Evaluation of different heuristics for accommodating asymmetric loss functions in regression”. In: *International Conference on Discovery Science*. Springer. 2017, pp. 67–81.

[57] Joaquin Vanschoren. “Meta-learning: A survey”. In: *arXiv preprint arXiv:1810.03548* (2018).

[58] Hal R Varian. “A Bayesian approach to real estate assessment”. In: *Studies in Bayesian econometric and statistics in Honor of Leonard J. Savage* (1975), pp. 195–208.

[59] Luisa Voller. “Literature Review on Automated Machine Learning (AutoML)”. In: ().

[60] Guodong Wang et al. “Student-teacher feature pyramid matching for unsupervised anomaly detection”. In: *arXiv preprint arXiv:2103.04257* (2021).

[61] Jie Yang et al. “Visual Anomaly Detection for Images: A Systematic Survey”. In: *Procedia Computer Science* 199 (2022). Publisher: Elsevier, pp. 471–478.

[62] Jiawei Yu et al. “Fastflow: Unsupervised anomaly detection and localization via 2d normalizing flows”. In: *arXiv preprint arXiv:2111.07677* (2021).

[63] Daochen Zha et al. “Meta-AAD: Active anomaly detection with deep reinforcement learning”. In: *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2020, pp. 771–780.

[64] Yue Zhao, Ryan A Rossi, and Leman Akoglu. “Automating outlier detection via meta-learning”. In: *arXiv preprint arXiv:2009.10606* (2020).

[65] Yue Zhao et al. “LSCP: Locally selective combination in parallel outlier ensembles”. In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 585–593.

[66] Jun Zhu, Nan Chen, and Weiwen Peng. “Estimation of bearing remaining useful life based on multiscale convolutional neural network”. In: *IEEE Transactions on Industrial Electronics* 66.4 (2018), pp. 3208–3216.

[67] Tiago Zonta et al. “Predictive maintenance in the Industry 4.0: A systematic literature review”. In: *Computers & Industrial Engineering* 150 (2020), p. 106889.